Mk 14

Micro Computer Training Manual

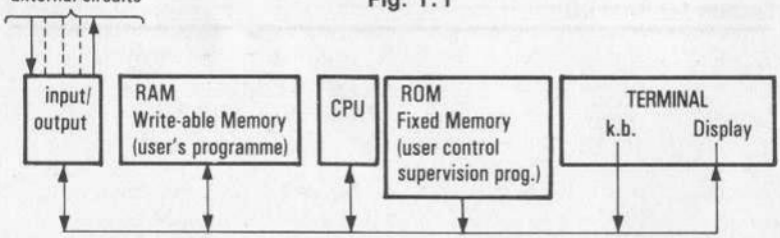
Contents

Part 1

1 Introduction to the kit

The MK14 comprises a full set of components to build up a completely functional computer.
When the unit has been correctly assembled only the connection of a suitable power source is needed for the display to light up and the user then finds that command and control of the unit is literally at his fingertips via the keyboard.
Having mastered the simple rules for operation of the keyboard and interpretation of the display, it is immediately possible to study the workings of the system and the computer's instructions, and experiment with elementary programming.
From this point the user can progress to the library of ready-written programmes available in Part II of this manual, and to programmes of his own invention. Because of the inherently enormous versatility of the digital computer it is hard to suggest any particular direction which the independent programmer may take. Arithmetic, logic, time measurement, complex decision making, learning ability, storage of data, receiving signals from other equipment and generating responses and stimuli can all be called upon.
Thus calculators, games, timers, controllers (domestic, laboratory, industrial), or combinations of these are all within the scope of the machine.

Fig. 1.1



Components of the kit include central processor, pre-programmed control memory, read-write memory, input/output circuits, the terminal section i.e. the keyboard and display, and interfacing to the terminal.
This line-up corresponds to the basic elements present in even the most sophisticated multi-million pound computer. Indeed the fundamental principles are identical. However, the user of the MK14 who wishes to understand and utilise these principles has the advantage of being able to follow in detail the action and inter-action of the constituent parts, which are normally inaccessible and invisible to the big computer operator.
Do not regard the MK14 as an electronics construction project. The MK14 is a computer, and computers are about software. It is the programme which brings the computer to life, and it is the programme which is capable of virtually infinite variation, adjustment and expansion.
Of course an understanding of the architecture of the machine and the functions of the separate integrated circuits is valuable to the user. But these aspects conform to a fairly standard pattern and the same straightforward set of interconnection rules regardless of the task or function the computer is performing.

2 The Manual - its objectives and uses

The MK14 is intended to bring practical computing to the widest possible range of users by achieving an absolute minimum cost. The wider the user spectrum, the wider, to be expected will be the variation of expertise the manual has to cater for; from the total novice, who wishes to learn the basic principles and requires thorough explanation of every aspect, to the experienced engineer who has immediate practical applications in view. Additionally, the needs of the beginner can be sub-divided into three parts:-

1. An informal step by step procedure to familiarise with the operation of the MK14. If this is arranged as an inter-active 'do' and 'observe' sequence, it becomes a comparatively painless method of getting a practical 'feel' for the computing process. Section 5.
2. A formal definition/description of the significant details of the microprocessor itself, i.e. its architecture and instruction set. Users of all levels are strongly recommended to study this section, (Section 0) at an early stage. It is supported by a programme of practical exercises aimed to precisely demonstrate the elemental functions of the device, and the framework inside which they operate. It is emphasised that to gain the most complete fluency in what are the basics of the whole subject is not merely well worth the effort but is essential to the user's convenience?
3. An explanation of the general principles of the digital processor, along with the associated notation and conventions. Section 0 this also breaks down into the joint aspects of hardware and software.

Clearly parts of the above will also prove useful to the knowledgable user who, however, will probably be able to skip the advice in section 3 on basic electronic assembly technique. The control part of this section contains information specifically pertinent to the MK14 and should be read by all.

Further sections to be referenced when the MK14 has been assembled, and the user has built up a working understanding, are those discussing programming techniques and methodology. From that point the applications examples of varying degrees of complexity and function, in Part II, should be possible for the reader to tackle.

3 Construction procedure Notes on soldering

The construction of the unit is a straightforward procedure consisting of inserting the components in the correct positions and soldering them in place. If this is done without error the system should become functional as soon as power is applied. To ensure that this happens without any hitches some recommendations and advice are offered. A step-by-step construction procedure with a diagram is laid down. An appendix to this section contains notes on soldering techniques.

Plug in socket option for integrated circuits

The I.C. components utilised in the MK14 are both robust and reliable. But accidents are possible—and should an I.C. be damaged either during construction or later, it's identification and replacement is made many orders easier if devices are mounted in sockets. Socket usage is therefore most strongly recommended, particularly where the user is concerned with computing rather than electronics. Science of Cambridge offer a MK14 rectification service specifying a component cost only replacement charge when the system in question is socket equipped.

Integrated Circuit Device Handling

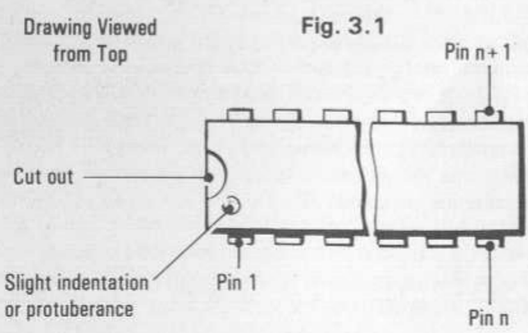
M.O.S. integrated circuits historically have gained a reputation for extreme vulnerability to damage from static electricity. Modern devices while not unbreakable embody a high degree of protection. This means that high static voltages will do no harm as long as the total energy dissipated is small and a practical rule of thumb is that if the environment is such that you yourself don't notice static shocks, neither will the I.C.
It is essential for the soldering iron to be earthed if I.C.'s are being soldered directly into the P.C. board. The earth must ground the soldering iron bit. This warning applies to any work carried out which might bring the soldering iron into contact with any I.C. pin.
Catastrophe is achievable with minimum trouble if certain components are fitted the wrong way round.

Component Orientation and I.C. Pin Numbering

Three types belonging to the kit must be oriented correctly. These are the I.C.'s, the electrolytic capacitors and the regulator.

(i) I.C's are oriented in relation to pin 1. Pin 1 can be identified by various means; fig. 3.1 illustrates some of these:-

Fig. 3.1



Pin 1 itself may bear a faint indentation or a slight difference from other pins. The remaining pins are numbered consecutively clockwise from Pin 1 viewing device as in Fig. 3.1.

Note position of type no. is **not** a reliable guide.

(ii) Electrolytic capacitors have a positive and a negative terminal. The positive terminal is indicated by a' + ' sign on the printed circuit. The capacitor may show a ' + ' sign or a bar marking by the positive terminal. The negative is also differentiated from the positive by being connected to the body of the device while the positive appears to emerge from an insulator.

(iii) The regulator has a chamfered edge and is otherwise assymmetrical- refer to assembly diagram.

Assembly Procedure

Equipment required—soldering iron, solder, side-cutters or wire snippers.

Step No.	Operation
1	Identify all resistors, bend leads according to diagram and place on layout diagram in appropriate positions.
2	Insert resistors into printed circuit and slightly bend leads at back of board so that resistors remain in place firmly against the P.C.
3	Solder resistors in place and cut surplus leads at back of printed circuit.
4	Re-check soldered joints and component positioning.
5	Identify all capacitors, bend leads according to diagram and place on layout diagram in appropriate positions.
6	Insert capacitors into printed circuit and slightly bend leads behind board so that capacitors remain in place firmly against the P.C.
7	Solder capacitors in place and cut surplus leads behind P.C.
8	Check soldered joints, component positions and orientation.
9	(If sockets are being used skip to step 14). Identify and place in position on diagram all I.C's with particular reference to orientation.
10	Insert I.C's into P.C. Note:- The I.C. pins will exhibit a degree of 'splay'. This allows the device to be retained in the P.C. mechanically after insertion so do not attempt to straighten, and use the following technique: place one line of pins so they just enter the board; using a suitable straight edged implement, press opposing row of pins until they enter the board; push component fully home.
11	Re-check device positioning and orientation with EXTREME care!

Step No.	Operation
12	Solder I.C's in place. It is not necessary to snip projecting pins.
13	Re-check all I.C. soldered joints. (skip to step 20)
14	Place appropriate sockets in position on diagram. See Fig. 3.3
15	Insert first or next socket in P.C. board. These components are not self retaining so invert the board and press onto a suitably resilient surface to keep socket firmly against the board while soldering.
16	Solder socket into position.
	(repeat steps 14-16 until all sockets are fitted)
17	Identify and place into position on diagram all I.C's with particular reference to orientation.
18	Transfer I.C's one-by-one to P.C. assembly and place in appropriate sockets.
19	Check all socket soldered joints.
20	Insert regulator and solder into position. See Fig. 3.4 (a).
21	Insert push button and solder into position. See Fig. 3.4 (b).
22	Mount keyboard. See Fig. 3.5.
23	Mount display. See Fig. 3.4 (c).
24	Ensure that all display interconnections are correctly aligned and inserted.
25	Solder display into position.
26	Re-check all soldering with special reference to dry joints and solder bridges as described in appendix on soldering technique.
27	(Optional but advisable). Forget the whole job for 24 hours.
28	Re-inspect the completed card by retracing the full assembly procedure and re-checking each aspect (component type, orientation and soldering) at each step. When the final inspection is satisfactorily completed proceed to section 4, Power Connect and Initial Operation.

Fig 3.4(a)

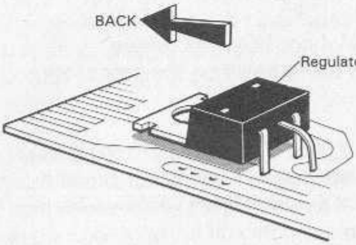


Fig 3.4(b)

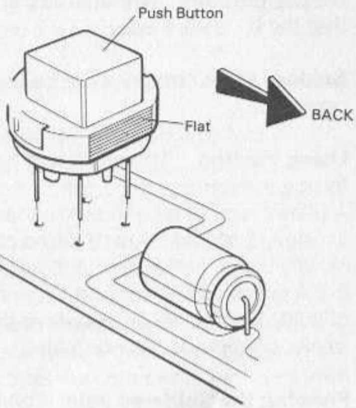


Fig 3.4(c)

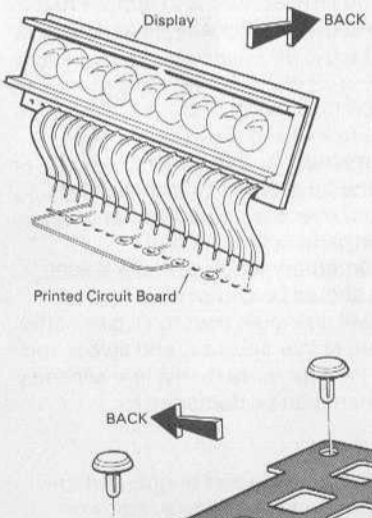
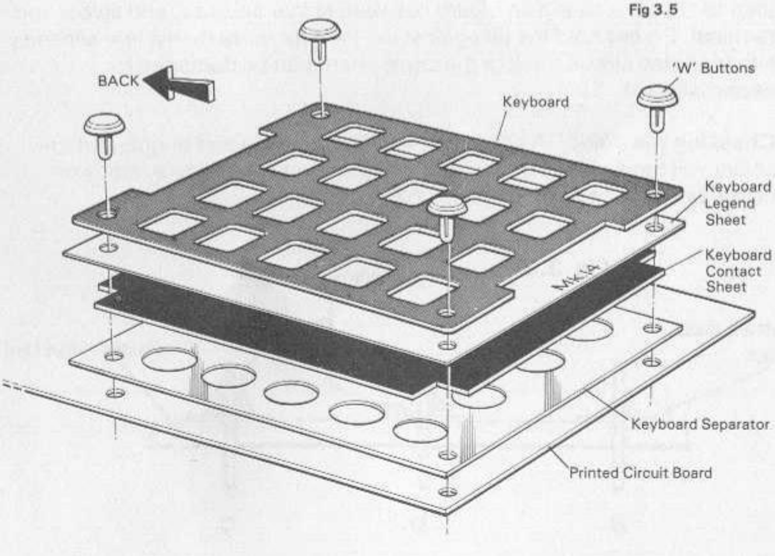


Fig 3.5



Appendix Soldering Technique

Poor soldering in the assembly of the MK14 could create severe difficulties for the constructor so here are a few notes on the essentials of the skill.

The Soldering Iron Ideally, for this job, a 15W/25W instrument should be used, with a bit tip small enough to place against any device pin and the printed circuit without fouling adjacent joints. IMPORTANT—ensure that the bit is earthed.

Solder resin cored should be used. Approx. 18 S.W.G. is most convenient.

Using the Iron The bit should be kept clean and be sufficiently hot to form good joints.
A plated type of bit can be cleaned in use by wiping on the dampened sponge (if available), or a damp cloth. A plain copper bit corrodes fairly rapidly in use and a clean flat working face can be maintained using an old file. A practical test for both cleanness and temperature is to apply a touch of solder to the bit, and observe that the solder melts instantly and runs freely, coating the working face.

Forming the Soldered Joint—with the bit thus 'wetted' place it into firm contact with **both** the component terminal and the printed circuit 'pad', being soldered together. Both parts must be adequately heated. Immediately apply solder to the face of the bit next to the joint. Solder should flow freely around the terminal and over the printed circuit pad. Withdraw the iron from the board in a perpendicular direction.
Take care not to 'swamp' the joint, a momentary touch with the solder should be sufficient. The whole process should be complete in one or two seconds. The freely flowing solder will distribute heat to all part of the joint to ensure a sound amalgam between solder and pad, and solder and terminal. Do not hold the bit against the joint for more than a few seconds either printed circuit track or the component can be damaged by excessive heat.

Checking the Joint A good joint will appear clean and bright, and the solder will have spread up the terminal and over the pad to a radius of about $\frac{1}{16}$ inch forming a profile as in Fig. 3.2(a).

Fig. 3.2

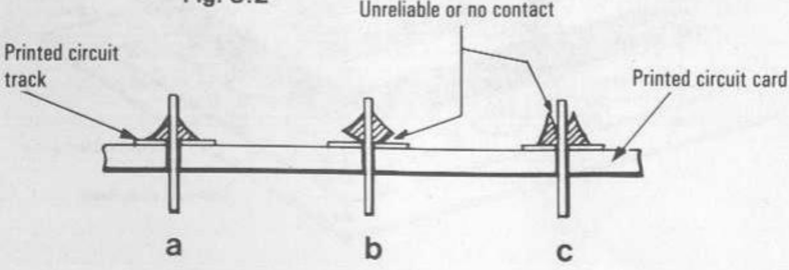


Fig 3.2 (b) and (c) show exaggerated profiles of unsuccessful joints. These can be caused by inadequate heating of one part, or the other, of the joint, due to the iron being too cool, or not having been in direct contact with both parts; or to the process being performed too quickly. An alternative cause might be contamination of the unsoldered surface.

Re-making the Joint Place the 'wetted' iron against the unsatisfactory joint, the solder will then be mostly drawn off. Re-solder the joint. If contamination is the problem it will usually be eliminated after further applications by the flux incorporated within the solder.

Solder 'Bridges'—can be formed between adjacent tracks on the printed circuit in various ways:—

(i) too cool an iron allowing the molten solder to be slightly tacky
(ii) excessive solder applied to the joint
(iii) bit moved away from the joint near the surface of the board instead of directly upwards

These bridges are sometimes extremely fine and hard to detect, but are easily removed by the tip of the cleaned soldering iron bit.

Solder Splashes—can also cause unwanted short circuits. Careless shaking of excess solder from the bit, or allowing a globule of solder to accumulate on the bit, must be avoided. Splashes are easily removed with the iron.

In summary, soldering is a minor manual skill which requires a little practise to develop. Adherence to the above notes will help a satisfactory result to be achieved.

4 Power Connect and Switch On

The MK14 operates from a 5V stabilised supply. The unit incorporates its own regulator, so the user has to provide a power source meeting the following requirements:—

Current consumption	Basic kit only —400mA
	+ RAM I/O option — + 50mA
	+ extra RAM option — + 30mA

Max I/P permitted voltage (including ripple) 35V

Min I/P permitted voltage (including ripple) 7V

Batteries or a mains driven power supply may be used. When using unregulated supplies ensure that ripple at the rated current does not exceed the I/P voltage limits.

If a power source having a mean output voltage greater than IOV has to be used, a heat sink must be fitted to the regulator. A piece of aluminium or copper, approx. 18 s.w.g., of about two square inches in area, bolted to the lug of the regulator should permit input voltages up to about 18V to be employed.

Alternatively a suitable resistor fitted in series with the supply can be used. To do this the value of the series resistor may be calculated as follows:-

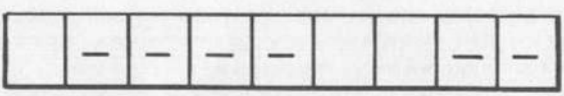
2 × (minimum value I/P voltage -7) Ω

Resistor dissipation will be 0.5W/Ω

Having selected a suitable power supply the most important precaution to observe is that of correct polarity. Connect power supply positive to regulator I/P and power supply negative to system ground.

Switch on.

Proper operation is indicated by the display showing this:—



Congratulations—now proceed to the section on usage familiarisation and learn to drive the MK14.

5 Usage Familiarisation

To help the user become accustomed to commanding and interrogating the MK14 an exercise consisting basically of a sequence of keyboard actions, with the expected display results, and an explanatory comment, is provided.
Readers who are not familiar with hexadecimal notation and data representation should refer to section 7.
It will be clear to those who have perused the section dealing with MK14 basic principles that to be able to utilise and understand the unit it is necessary firstly to have the facility to look at the contents of locations in memory I/O and registers in the CPU, and secondly to have the facility to change that information content if desired.
The following shows how the monitor programme held in fixed memory enables this to be done.

Operator Action	Display		Comment
			Examining MK14 Memory
Switch on	- - - - - -		The left hand group of four characters is called the address field, the right hand group is the data field. Dashes indicate that the MK14 is waiting for a GO or a MEM command.
MEM	0000	08	The contents of memory location zero is displayed in the data field.
MEM	0001	90	Next address in sequence is displayed, and the data at that address.
MEM	0002	1D	Address again incremented by one, and the data at the new address is displayed.
MEM	0003	C2	Next address and contents are displayed

The user is actually accessing the beginning of the monitor programme itself. The items of data 08, 90, 1D, C2 are the first four instructions in the monitor programme.
It is suggested that for practise a list of twenty or thirty of these is made out and the appropriate instruction nmemonics be filled in against them from the list of instructions in Section 9. Additionally, this memory scanning procedure offers an introduction to the hexadecimal numbering method used by the addressing system, as each MEM depression adds one to the address field display.

Operator Action	Display	Comment
		Loading MK14 Memory
MEM	XXXX XX	note:—symbol X indicates when digit value is unpredictable or un-important.
0	0000 XX	First digit is entered to L & D address field, higher digits become zero.
F	000F XX	Second address digit keyed enters display from right.
1	00F1 XX	Third address digit keyed enters display from right.
2	0F12 XX	This is first address in RAM available to the user (basic version of kit).
TERM	0F12 XX	TERM enters displayed address and prepares for operator to load data.
1	0F12 01	Memory data has been keyed but is not yet placed in RAM.
TERM	0F12 01	Data is now placed in RAM
MEM	0F13 XX	Address is incremented.
TERM	0F13 XX	New address is entered and unit waits for memory data input.
1	0F13 01	New data.
1	0F13 11	is keyed
TERM	0F13 11	and placed in memory
MEM	0F14 XX	Data
TERM	0F14 XX	is
22	0F14 22	loaded
TERM	0F14 22	into
MEM	0F15 XX	successive
TERM	0F15 XX	locations
33	0F15 33	
TERM	0F15 33	
MEM	0F16 XX	

Operator Action	Display		Comment
44	OF16	44	
TERM	OF16	44	
OF12	OF12	01	Enter original memory address and
MEM	OF13	11	check that data
MEM	OF14	22	remains as
MEM	OF15	33	was
MEM	OF16	44	loaded.

Switch power off and on again. Re-check contents of above locations. Note that loss of power destroys read-write memory contents. Repeat power off/on and re-check same locations several times—it is expected that RAM contents will be predominately zero, and tend to switch on in same condition each time. This effect is not reliable.

Operator Action	Display		Comment
MEM	XXXX	XX	**Enter a very small programme**
OF12TERM	OF12	XX	It consists of one instruction JMP-2 (9OFE in
90	OF12	90	machine code). 90 represents JUMP programme
TERM MEM	OF13	XX	counter relative. FE represents −2, the direction
TERM FE	OF13	FE	of the jump.
TERM	OF13	FE	
ABORT	-----	-	
GO	OF13	--	Prepare to start user programme (TERM at this point would start execution from OF12).
OF12	OF12	--	Enter start address.
TERM	BLANK		Commence execution. The display becomes blank, indicating that CPU has entered user programme, and remains blank.

We have created the most elementary possible programme—one that loops round itself. There is only one escape—RESET which will force the CPU to return to location 1.

RESET	----	--	Reset does not affect memory the instruction JMP−2 is still lurking to trap the user.

6 Basic Principles of the MK14

Essentially the MK14 operates on exactly the same principles as do all digital computers. The 'brain' of the MK14 is a SC/MP micro-processor, and therefore aspects of the SC/MP will be used to illustrate the following explanation. However the principles involved are equally valid for a huge machine from International Computers down to pocket calculators. Moreover, these principles can be stated quite briefly, and are essentially very simple.

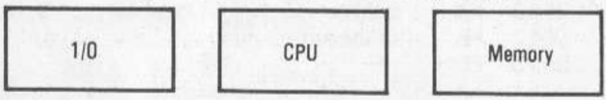
'Stored Programme' Principle

The SC/MP CPU (Central Processing Unit) tends to be regarded as the centre-piece because it is the 'clever' component—and so it is. But by itself it can do nothing. The CPU shows its paces when it is given INSTRUCTIONS. It can obey a wide range of different orders and perform many complex digital operations. This sequence of instructions is termed thePROGRAMME, and is STORED in the MEMORY element of the system. Since these instructions consist of manipulation and movement if data, in addition to telling the CPU what to do, the stored programme contains information values for the CPU to work on, and tells the CPU where to get information, and where to put results.

Three Element System

By themselves the two fundamental elements CPU and MEMORY can perform wondrous things—all of which would be totally useless, since no information can be input from the outside world and no results can be returned to the user. Consequently a third element has to be incorporated—the INPUT/OUTPUT (I/O) section.

Fig. 6.1 The Three Element System



These three areas constitute the HARDWARE of the system, so called because however you may use or apply the MK14, these basic structures remain the same.

Independence of Software (Stored Programme) and Hardware

As with the other hardware, whatever particular instruction sequence is present within the memory at any one time, the basic structure of the memory element itself is unaltered.

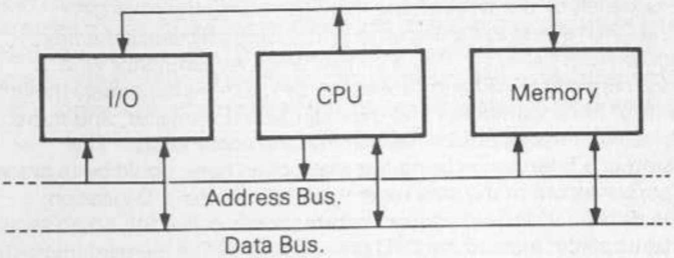
It is this factor which gives the MK14 its great versatility: by connecting up its I/O and entering an appropriate programme into its memory it can perform any digital function that can be contained within the memory and I/O size.

Random Access Memory (RAM)

Further, when the memory in question consists of a read **and write** element (RAM), in contrast to read **only** memory (ROM), this flexibility is enhanced, as programme alterations, from minor modifications, to completely different functions, can be made with maximum convenience.

Interconnection of Basic Elements
Element inter-connection is standardised as are the elements themselves. Three basic signal paths, ADDRESS BUS (ABUS), DATA BUS (DBUS) and CONTROL BUS, are required.

Fig. 6.2 Interconnections of Three Element System



These buses are, of course, multi-line. In the MK14 the Abus = 12 lines, Dbus = 8 lines and Control bus = 3lines. Expansion of memory or 1/O simply requires connection of additional elements to this basic bus structure.

MK14 System Operation
Consider the MK14 with power on and the RESET signal applied to the SC/MP. This forces all data inside the CPU to zero and prevents CPU operation.
When the RESET is released the CPU will place the address of the first instruction on the Abus and indicate that an address is present by a signal on the ADDRESS STROBE (NADS) line which is within the control bus.
The memory will then respond by placing the first instruction on the Dbus.
The CPU accepts this information and signals a READ STROBE (NRDS) via a line within the control bus.
The CPU now examines this instruction which we will define as a no-operation, (instructions are normally referred to by abbreviations called NMEMONICS, the nmemonic fof this one is NOP).
In obedience the CPU does nothing for one instruction period and then sends out the address of the second instruction. The memory duly responds with a Load Immediate (LDI). The CPU interprets this to mean that the information in the next position, in sequence, in memory will not be an instruction but an item of data which it must place into its own main register (ACCUMULATOR). so the CPU puts out the next address in sequence, and when the memory responds with data, then obeys the instruction.
The CPU now addresses the next position (LOCATION) in memory and fetches another instruction—store (ST). This will cause the CPU to place the data in the accumulator back on the Dbus and generate a WRITE STROBE (NWRDS) via the control bus. (The programme's intention here is to set output lines in the 1/O element to a pre-determined value).
Before executing the store instruction the CPU addresses the next sequential location in memory, and fetches the data contained in it. The purpose of this data word is to provide addressing information needed, at this point, by the CPU.
So far, consecutive addresses have been generated by the CPU in order to fetch instructions or data from memory. In order to carry out the store

instruction the CPU must generate a different address, with no particular relationship to the instruction address itself, i.e. an address in the I/O region.
The CPU now constructs this address using the aforementioned data word and outputs it to the Abus. The I/O element recognises the address and accepts the data appearing on the Dbus (from the CPU accumulator), when signalled by the write strobe (NWRDS), also from the CPU.
Now the CPU reverts to consecutive addressing and seeks the next instruction from memory. This is an Exchange Accumulator with Extension register (XAE) and causes the CPU to simultaneously move the contents of the accumulator into the extension (E) register, and move the contents of the extension register into the accumulator. The programmer's intention in using this instruction here, could be to preserve a temporary record of the data recently written to the I/O location.
No new data or additional address information is called for, so no second fetch takes place. Instead the CPU proceeds to derive the next instruction in sequence.
For the sake of this illustration we will look at a type of instruction which is essential to the CPU's ability to exhibit intelligence.
This is the jump (JMP) instruction, and causes the CPU to depart from the sequential mode of memory accessing and 'jump' to some other location from which to continue programme execution.
The JMP will be back to the first location.
A JMP instruction requires a second data word, known as the DISPLACEMENT to define the distance and direction of the jump.
Examining the memory I/O contents map, Fig 6.3, shows location 0 to be seven places back from the JMP displacement which therefore must have a numerical value equivalent to −7. (Detail elsewhere in this manual will show that this value is not precisely correct, but it is valid as an example).
The instruction fetched after executing the JMP will be the NOP again.
In fact the sequence of five instructions will now be re-iterated continually.
The programme has succumbed to a common bug—an endless loop, in which for the time being we will leave it.

Fig. 6.3 Map of Memory Location Contents.

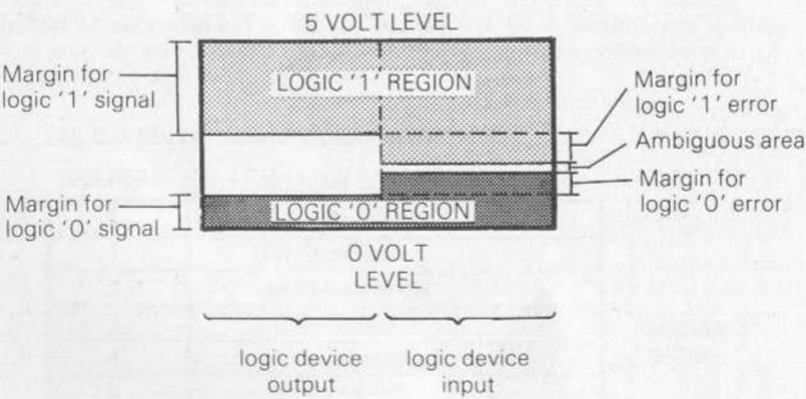
LOCATION No.	LOCATION CONTENTS	
0	NOP (instruction)	MEMORY REGION
1	LDI (instruction)	
2	data (for use by LDI)	
3	ST (instruction)	
4	address information (for use by ST)	
5	XAE (instruction)	
6	JMP (instruction)	
7	−7 (displacement for JMP)	
Formed by CPU using data in loc. 4	Initially undefined—after 3 becomes same as loc. 2	I/O REGION

This brief review of a typical sequence of MK14 internal operations has emphasised several major points. All programme control and data derives from the memory and 1/O. All programme execution is performed by the CPU which can generate an address to any location in memory and 1/O, and can control data movement to or from memory and 1/O.
Some instructions involve a single address cycle and are executed within the CPU entirely. Other instructions involve a second address cycle to fetch an item of data, and sometimes a third address cycle is also needed. For the sake of simplicity this outline has deliberately avoided any detail concerning the nature of the instruction/data, and the mechanics of the system. These subjects are dealt with in greater depth in sections 5 and 7.

7 MK14 Language-Binary and Hexadecimal

Discussion of the MK14 in this handbook so far has referred to various categories of data without specifying the physical nature of that data. This approach avoids the necessity of introducing too many possibly unfamiliar concepts at once while explaining other aspects of the workings of the system.
This section, then, gives electrical reality to the abstract forms of information such as address, data, etc., which the computer has to understand and deal with.
Binary Digit Computers use the most fundamental unit of information that exists—the binary digit or BIT—the bit is quite irreducible and fundamental. It has two values only, usually referred to as '0' and '1'. Human beings utilise a numbering system possessing ten digits and a vocabulary containing many thousands of words, but the computer depends on the basic bit.
However, the bit is readily convertible into an electrical signal. Five volts is by far the most widely used supply line standard for electronic logic systems. Thus a zero volt (ground) level represents '0', and a positive five volt level represents '1'. Note that the SC/MP CPU follows this convention which is known as positive logic; negative logic convention determines inverse conditions, i.e. 5V = '0', 0V = '1'.
Logic Signal Voltage Limits For practical purposes margins must be provided on these signal levels to allow for logic device and system tolerances. Fig. 7.1 shows those margins.

Fig. 7.1

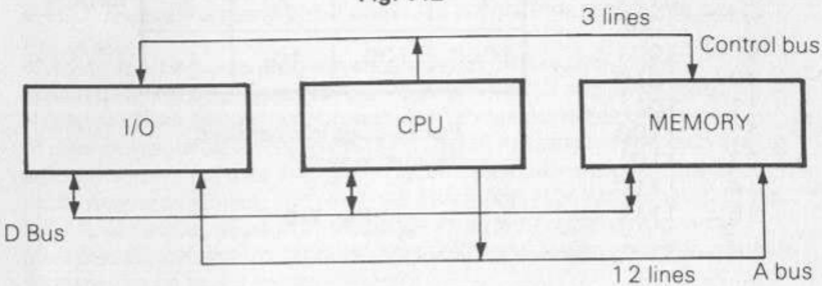


'0's and '1's Terminology Many of the manipulation rules for '0's and '1's are rooted in philosophical logic, consequently terms like 'true' and 'false' are often used for logic signals, and a 'truth table' shows all combinations of logic values relating to a particular configuration. The

control engineer may find 'on' and 'off' more appropriate to his application, while an electronic technician will speak of 'high' and 'low', and to a mathematician they can represent literally the numerals one and zero.

Using Bits in the MK14 The two state signal may appear far too limited for the complex operations of a computer, but consider again the basic three element system and it's communication bus.

Fig. 7.2



The data bus for example comprises eight lines. Using each line separately permits eight conditions to be signalled. However, eight lines possessing two states each, yield 256(2^8) combinations, and the A bus can yield 4096 combinations.

A group or WORD of eight bits is termed a BYTE

Decoding In order to tap the information potential implied by the use of combinations, the elements in the MK14 all possess the ability to DECODE bit combinations. Thus when the CPU generates an address, the memory I/O element is able to select one out of 4096 locations. Similarly, when the CPU fetches an instruction from memory it obeys one out of 128 possible orders.

Apart from instructions, depending on context, the CPU treats information on the data bus sometimes as a numerical value, or sometimes simply as an arbitrary bit pattern, thereby further expanding data bus information capacity.

Bits as Numbers When grouped into a WORD the humble bit is an excellent medium for expressing numerical quantities. A simple set of rules exist for basic arithmetic operations on binary numbers, which although they lead to statements such as 1 + 1 = 10, or 2_{10} and 2_{10} make 100_2, they can be executed easily by the ALU (Arithmetic and Logic Unit) within the CPU. Note that the subscripts indicate the base of the subscripted numbers.

Binary Numbers The table below compares the decimal values 0—15 with the equivalent binary notation.

Decimal	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

Most significant digit (MSD) → 8; Least significant digit (LSD) → 1

8	4	2	1	BINARY
1000_s	100_s	10_s	1_s	DECIMAL

Place values in binary and decimal systems

Fig. 7.3

The binary pattern is self evident, and it can also be seen how place value of a binary number compares with that in the decimal system.
Expressed in a different way, moving a binary number digit one place to the left doubles its value, while the same operation on a decimal digit multiplies its value by ten.
The Binary pattern is self evident, and it can also be seen how place value of a binary number compares with that in the decimal system.
Binary Addition—requires the implementation of four rules:—

$0+0=0$

$0+1$ or $1+0=1$

$1+1=1$ with carry (to next higher digit)

$1+1+$ carry (from next lower digit) $=1$ with carry (to next higher digit)

```
Example:—  1110110
          +1010101
          --------
          11001011
          111  1     ←carry indications
```

Binary Subtraction

$0-0=0$

$1-1=0$

$1-0=1$

$0-1=1$ with borrow (from next higher digit)

$0-1-$borrow (from next lower digit) $=1$ with borrow (from next higher digit)

```
              0 1        0 1 1       0 10 1   ← borrow indications
Examples:—    1̸01        1̸0̸0        110
             -010       -001        -011
              011        011         011
```

8 Program Notes

At the point the reader is likely to be considering the application programmes in Part II and perhaps devising some software of his own. This section examines the manner in which a programme is written and set out, the planning and preparation of a programme, and some basic techniques.

When embarking on a programme two main factors should be considered, they are: (i) hardware requirements, (ii) sequence plan.

Hardware Requirements An assessment should be made of the amount of memory required for the instruction part of the programme, and the amount needed for data storage. In a dedicated micro-processor system these will occupy fixed, and read-write memory areas respectively. In the MK14, of course, all parts of the programme will reside in read-write memory, simplifying the programmers task considerably, since local pools for data can be set up indiscriminately.

However, even in the MK14 more care must be given to the allocation of memory space for common groups of data and for input/output needs. The SC/MP C.P.U. offers a certain amount of on-chip input/output in terms of three latched flags, two sense inputs, and the serial in/serial out terminals. So the designer must decide if these are more appropriate to his application than the memory mapped I/O available in the RAMIO option.

Memory Map A useful aid in this part of the process is the memory map diagram which gives a spatial representation to the memory and I/O resources the programmer has at his disposal. Fig. 8.1 shows the MK14 memory map including both add-in options

Standard RAM →	RAM
	RAMIO
	DISPLAY
	RAMIO
Optional RAM →	RAM
	RAMIO
	DISPLAY
256 io locations →	RAMIO
	MONITOR
	MONITOR
	MONITOR
512 locations →	MONITOR

Fig. 8.1

The map displays the memory as a column of 4K locations, (in this case each of eight bits), with location zero at the base and addresses ascending upwards.

The reader may be surprised that various sections of memory appear to reside in several areas at once.

For example the monitor is repeated four times in the lower 2K block. Note also that the monitor will only operate correctly if executed in the lowest section, as only this section has the proper relationship to the RAM at the top.

These multiple appearances of memory blocks are due to partial address decoding technique employed to minimise decode components.

The map readily indicates that a CPU memory pointer (which can permit access to a block of 256 I/O locations) set to 0900_{16} would give the programme a stepping stone into the display O/P or the RAMIO facilities.

Flow Chart The flow chart provides a graphical representation of the sequence plan. A processor is essentially a sequential machine and the flow chart enforces this discipline. Fig. 8.2 is a very simple example of a programme to count 100 pulses appearing at an input. Three symbols are used (i) the **circle** for entry or exit points (ii) the **rectangle** for programme operations (iii) the **diamond** for programme decisions.

A flow chart should always be prepared when constructing a programme. Each block is a convenient summary of what may be quite a large number of instructions. Of particular value is the overview provided of the paths arising from various combinations of branch decisions.

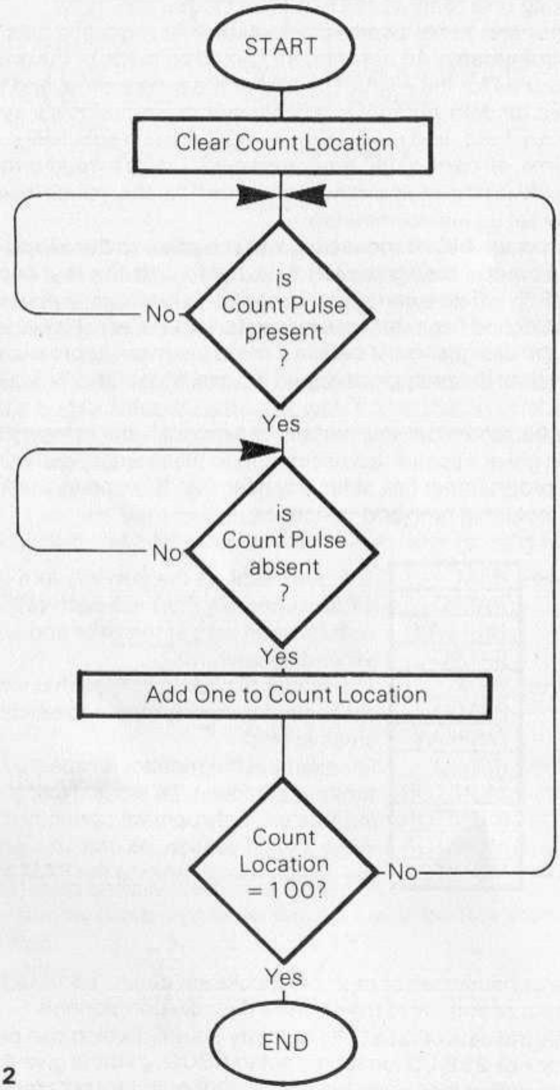


Fig. 8.2

The flow chart can reveal wasteful repetition or logical anomalies, and ensures that like a good story, the programme starts at the beginning, progresses through the middle, and comes to a satisfactory end.

Programme Notation There is a well established convention and format for writing down a programme listing. We will examine two lines extracted from the MK14 monitor programme itself in order to define the various functions of the notation.

```
(a)    (b)    (c)
112    0003   GOOUT:
              (d)    (e)    (f)    (g)
113    0003   C2OE   LD     ADH    (2)      ;GET GO ADDRESS
```

a) Line Number. All lines in the listing are consecutively numbered for reference.

b) Location Counter. The current value of the location counter (programme counter in the CPU) is shown wherever it is relevant e.g. when the line contains a programme instruction or address label.

c) Symbolic Address Label. This is followed by a colon. Entry points to sub-sections of programme can be labelled with meaningful abbreviations making the programme easier to follow manually e.g. at some other place in the programme a JUMP TO 'GOOUT' might occur. Automatic assemblers create an internal list of labels and calculate the jump distances.
However the MK14 user must do it the hard way.

d) Machine Code. The actual code in the memory is shown here. As it is a two byte instruction the first two hexadecimal digits C2 are in location 3 and OE is in location 4.

e) Nmemonic LD is the nmemonic for LOAD. This is the instruction represented by C2 in machine code.

f) Displacement. ADH is another label, in this case for a data value. Note that a table is provided in alpha-numeric order at the end of the listing, of all symbols and their values.

g) Pointer Designation. Define the pointer to be referenced by this instruction.

h) Comment. All text following the semi-colon is explanatory material to explain the purpose of the instruction or part of programme.

9 Architecture and Instruction Set

The SC/MP microprocessor contains seven registers which are accessible to the programmer. The 8-bit accumulator, or AC, is used in all operations. In addition there is an 8-bit extension register, E, which can be used as the second operand in some instructions, as a temporary store, as the displacement for indexed addressing, or in serial input/output. The 8-bit status register holds an assortment of single-bit flags and inputs:

SC/MP Status Register

7	6	5	4	3	2	1	0
CY/L	OV	S_B	S_A	IE	F_2	F_1	F_0

Flags	Description
F_0-F_2	User assigned flags 0 through 2.
IE	Interrupt enable, cleared by interrupt.
S_A,S_B	Read-only sense inputs. If IE = 1, S_A is interrupt input.
OV	Overflow, set or reset by arithmetic operations.
CY/L	Carry/Link, set or reset by arithmetic operations or rotate with Link.

The program counter, or PC, is a 16-bit register which contains the address of the instruction being executed. Finally there are three 16-bit pointer registers, P1, P2, and P3, which are normally used to hold addresses. P3 doubles as an interrupt vector.

Addressing Memory

All memory addressing is specified relative to the PC or one of the pointer registers. Addressing relative to the pointer registers is called indexed addressing. The basic op-codes given in the tables below are for PC-relative addressing. To get the codes for indexed addressing the number of the pointer should be added to the code. The second byte of the instruction contains a displacement, or disp., which gets added to the value in the PC or pointer register to give the effective address, or EA, for the instruction. This disp. is treated as a signed twos-complement binary number, so that displacements of from -128_{10} to $+127_{10}$ can be obtained. Thus PC-relative addressing provides access to locations within about 128 bytes of the instruction; with indexed addressing any location in memory can be addressed.

Instruction Set

7 . . . 3	2	1 0
Op	m	ptr

byte 1

7 0
disp

byte 2

Memory Reference

Mnemonic	Description	Operation	Op Code Base
LD	Load	(AC)←(EA)	C000
ST	Store	(EA)←(AC)	C800
AND	AND	(AC)←(AC) ∧ (EA)	D000
OR	OR	(AC)←(AC) ∨ (EA)	D800
XOR	Exclusive-OR	(AC)←(AC) ∨ (EA)	E000
DAD	Decimal Add	$(AC) \leftarrow (AC)_{10} + (EA)_{10} + (CY/L);(CY/L)$	E800
ADD	Add	(AC)←(AC) + (EA) + (CY/L);(CY/L),(OV)	F000
CAD	Complement and Add	(AC)←(AC) + ¬(EA) + (CY/L);(CY/L),(OV)	F800

Base Code Modifier

Op Code = Base + m + ptr + disp

Address Mode	m	ptr	disp	Effective Address
PC-relative	0000	0000	00xx	EA = (PC) + disp
Indexed	0000	0100 0200 0300	00xx	EA = (ptr) + disp
Auto-indexed	0400	0100 0200 0300	00xx	If disp⩾0, EA = (ptr) If disp<0, EA = (ptr) + disp

xx = −128 to +127

Note: If disp = −128, then (E) is substituted for disp in calculating EA.

The operands for the memory reference instructions are the AC and a memory address.

With these eight instructions the auto-indexed mode of addressing is available; the code is obtained by adding 4 to the code for indexed addressing. If the displacement is positive it is added to the contents of the specified pointer register **after** the contents of the effective address have been fetched or stored. If the displacement is negative it is added to the contents of the pointer register **before** the operation is carried out. This asymmetry makes it possible to implement up to three stacks in memory; values can be pushed onto the stacks or pulled from them with single auto-indexed instructions. Auto-indexed instructions can also be used to add constants to the pointer registers where 16-bit counters are needed.

A special variant of indexed or auto-indexed addressing is provided when the displacement is specified as X'80. In this case it is the contents of the extension register which are added to the specified pointer register to give the effective address. The extension register can thus be used simultaneously as a counter and as an offset to index a table in memory.

For binary addition the 'add' instruction should be preceded by an instruction to clear the CY/L. For binary subtraction the 'complement and add' instruction is used, having first **set** the CY/L. Binary-coded-decimal arithmetic is automatically handled by the 'decimal add' instruction.

7 0	7 0
Op	data
byte 1	byte 2

Immediate

Mnemonic	Description	Operation	Op Code Base
LDI	Load Immediate	(AC)←data	C400
ANI	AND Immediate	(AC)←(AC) A data	D400
ORI	OR Immediate	(AC)←(AC) V data	DC00
XRI	Exclusive-OR Immediate	(AC)←(AC) V data	E400
DAI	Decimal Add Immediate	$(AC)\leftarrow(AC)_{10} + data_{10} + (CY/L);(CY/L)$	EC00
ADI	Add Immediate	(AC)←(AC) + data + (CY/L);(CY/L),(OV)	F400
CAI	Complement and Add Immediate	(AC)←(AC) + ^data + (CY/L);(CY/L),(OV)	Fc00

Base Code Modifier
Op Code = Base + data

the immediate instructions specify the actual data for the operation in the second byte of the instruction.

7 0
Op

Extension Register

Mnemonic	Description	Operation	Op Code
LDE	Load AC from Extension	(AC)←(E)	40
XAE	Exchange AC and Ext.	(AC)↔(E)	01
ANE	AND Extension	(AC)←(AC) A (E)	50
ORE	OR Extension	(AC)←(AC) V (E)	58
XRE	Exclusive-OR Extension	(AC)←(AC) V (E)	60
DAE	Decimal Add Extension	$(AC)\leftarrow(AC)_{10} + (E)_{10} + (CY/L), (CY/L)$	68
ADE	Add Extension	(AC)←(AC) + (E) + (CY/L); (CY/L), (OV)	70
CAE	Complement and Add Extension	(AC)←(AC) + ~ (E) + (CY/L); (CY/L), (OV)	78

The extension register can replace the memory address as one operand in the above two-operand instructions. The extension register can be loaded by means of the XAE instruction.

7 . . . 2	1 0	7 0
Op	ptr	disp
byte 1		byte 2

Memory Increment/Decrement

Mnemonic	Description	Operation	Op Code Base
ILD	Increment and Load	(AC), (EA)←(EA) + 1	A800
DLD	Decrement and Load	(AC), (EA)←(EA)−1	B800
		Note: The processor retains control of the input/output bus between the data read and write operations.	

Base Code Modifier		
Op Code = Base + ptr + disp		
ptr	disp	Effective Address
0100 0200 0300	00xx	EA = (ptr) + disp
xx = −128 to +127		

The 'decrement and load' instruction decrements the contents of the memory location specified by the second byte, leaving the result in the accumulator. This provides a neat way of performing a set of instructions several times. For example:

```
        LDI     9
        ST      COUNT
LOOP:   ....
        ....
        DLD     COUNT
        JNZ     LOOP
```

will execute the instructions within the loop 9 times before continuing. Both this and the similar 'increment and load' instruction leave the CY/L unchanged so that multibyte arithmetic or shifts can be performed with a single loop.

Transfer

7 . . . 2	1 0		7 0
Op	ptr		disp
byte 1			byte 2

Mnemonic	Description	Operation	Op Code Base
JMP	Jump	(PC)←EA	9000
JP	Jump if Positive	If (AC)≥0, (PC)←EA	9400
JZ	Jump if Zero	If (AC) = 0, (PC)←EA	9800
JNZ	Jump if Not Zero	If (AC) ≠ 0, (PC)←EA	9C00

Base Code Modifier

Op Code = Base + ptr + disp

Address Mode	ptr	disp	Effective Address
PC-relative	0000	00xx	EA = (PC) + disp
Indexed	0100 0200 0300	00xx	EA = (ptr) + disp

xx = −128 to +127

Transfer of control is provided by the jump instructions which, as with memory addressing, are either PC-relative or relative to one of the pointer registers. Three conditional jumps provide a way of testing the value of the accumulator. 'Jump if positive' gives a jump if the top bit of the AC is zero. The CY/L can be tested with:

```
CSA              ;Copy status to AC
JP      NOCYL    ;CY/L is top of bit status
```

which gives a jump if the CY/L bit is clear.

Pointer Register Move

7 . . . 2	1 0
Op	ptr

Mnemonic	Descripton	operation	Op Code Base
XPAL	Exchange Pointer Low	(AC)↔(PTR$_{7:0}$)	30
XPAH	Exchange Pointer High	(AC)↔(PTR$_{15:8}$)	34
XPPC	Exchange Pointer with PC	(PC)↔(PTR)	3C

Base Code Modifier

Op Code = Base + ptr

The XPAL and XPAH instructions are used to set up the pointer registers, or to test their contents. For example, to set up P3 to contain X'1234 the following instructions are used:

```
LDI   X'12
XPAH   3
LDI   X'34
XPAL   3
```

The XPPC instruction is used for transfer of control when the point of transfer must be saved, such as in a subroutine call. The instruction exchanges the specified pointer register with the program counter, causing a jump. The value of the program counter is thus saved in the register, and a second XPPC will return control to the calling point. For example, if after the sequence above an XPPC 3 was executed the next instruction executed would be the one at X'1235. Note that this is one beyond the address that was in P3 since the PC is incremented before each instruction. P3 is used by the MK14 monitor to transfer control to the user's program, and an XPPC 3 in the user's program can therefore be used to get back to the monitor provided that P3 has not been altered.

Shift Rotate Serial I/O

7 0
Op

Mnemonic	Description	Operation	Op Code
SIO	Serial Input/Output	$(E_i) \rightarrow (E_{i-1})$, $SIN \rightarrow (E_7)$, $(E_0) \rightarrow SOUT$	19
SR	Shift Right	$(AC_i) \rightarrow (AC_{i-1})$, $0 \rightarrow (AC_7)$	1C
SRL	Shift Right with Link	$(AC_i) \rightarrow (AC_{i-1})$, $CY/L) \rightarrow (AC_7)$	1D
RR	Rotate Right	$(AC_i) \rightarrow (AC_{i-1})$, $(AC_0) \rightarrow (AC_7)$	1E
RRL	Rotate Right with Link	$(AC_i) \rightarrow (AC_{i-1})$, $(AC_0) \rightarrow (CY/L) \rightarrow (AC_7)$	1F

The SIO instruction simultaneously shifts the SIN input into the top bit of the extension register, the bottom bit of the extension register going to the SOUT output; it can therefore form the basis of a simple program to transfer data along a two-way serial line. The shift and rotate with link make possible multibyte shifts or rotates.

Double Byte Miscellaneous

7 0	7 0
Op	Disp
byte	byte 2

Mnemonic	Description	Operation	Op Code Base
DLY	Delay	count AC to -1, delay $= 13 + 2(AC) + 2$ disp $+ 2^9$ disp microcycles	8F00

Base Code Modifier
Op Code = Base + disp

The delay instruction gives a delay of from 13 to 131593 microcycles which can be specified in steps of 2 microcycles by the contents of the AC and the second byte of the instruction.
Note that the AC will contain X'FF after the instruction.

7   0
Op

Single-Byte Miscellaneous

Mnemonic	Description	Operation	Op Code
HALT	Halt	Pulse H-flag	00
CCL	Clear Carry/Link	(CY/L)←0	02
SCL	Set Carry/Link	(CY/L)←1	03
DINT	Disabled Interrupt	(IE)←0	04
IEN	Enable Interrupt	(IE)←1	05
CSA	Copy Status to AC	(AC)←(SR)	06
CAS	Copy AC to Status	(SR)←(AC)	07
NOP	No Operation	(PC)←(PC) + 1	08

The remaining instructions provide access to the status register, and to the IE and CY/L bits therein. The HALT instruction will act as a NOP in the MK14 kit unless extra logic is added to detect the H-flag at NADS time, in which case it could be used as an extra output.

Mnemonic Index of Instructions

Mnemonic	Opcode	Read Cycles	Write Cycles	Total Microcycles
ADD	F0	3	0	19
ADE	70	1	0	7
ADI	F4	2	0	11
AND	D0	3	0	18
ANE	50	1	0	6
ANI	D4	2	0	10
CAD	F8	3	0	20
CAE	78	1	0	8
CAI	FC	2	0	12
CAS	07	1	0	6
CCI	02	1	0	5
CSA	06	1	0	5
DAD	E8	3	0	23
DAE	68	1	0	11
DAI	EC	2	0	15
DINT	04	1	0	6
DLD	B8	3	1	22
DLY	8F	2	0	13-131593

Mnemonic	Opcode	Read Cycles	Write Cycles	Total Microcycles
HALT	00	2	0	8
IEN	05	1	0	6
ILD	A8	3	1	22
JMP	90	2	0	11
JNZ	9C	2	0	9, 11 for Jump
JP	94	2	0	9, 11 for Jump
JZ	98	2	0	9, 11 for Jump
LD	C0	3	0	18
LDE	40	1	0	6
LDI	C4	2	0	10
NOP	08	1	0	5
OR	D8	3	0	18
ORE	58	1	0	6
ORI	DC	2	0	10
RR	1E	1	0	5
RRL	1F	1	0	5
SCL	03	1	0	5
SIO	19	1	0	5
SR	1C	1	0	5
SRL	1D	1	0	5
ST	C8	2	1	18
XAE	01	1	0	7
XOR	E0	3	0	18
XPAH	34	1	0	8
XPAL	30	1	0	8
XPPC	3C	1	0	7
XRE	60	1	0	6
XRI	E4	2	0	10

Program Listings

The application program listings at the end of this manual are given in a symbolic form known as 'assembler listings'. The op codes are represented by mnemonic names of from 2 to 4 letters, with the operands specified as shown:

```
LD   disp            ;PC-relative addressing
LD   disp (ptr)      ;Indexed addressing
LD   @disp (ptr)     ;Auto-indexed addressing
```

Constants and addresses are also sometimes represented by names of up to six letters; these names stand for the same value throughout the program, and are given that value either in an assignment statement, or by virtue of their appearing as a label to a line in the program. Some conventions used in these listings are shown below:

Statements **Directive**

Assembler Format	Function
.END (address)	Signifies physical end of source pprogram.
.BYTE exp (,exp...)	Generates 8-bit (single-byte) data in successive memory locations.
.DBYTE exp (,exp,...)	Generates 16-bit (double-byte) data in successive memory locations.

Statements **Assignment**

LABEL:	SYMBOL = EXPRESSION	;Symbol is assigned ;value of expression
	. = 20	;Set location counter ;to 20
TABLE:	. = . + 10	;Reserve 10 locations for table

10 RAM I/O

A socket is provided on the MK14 to accept the 40 pin RAM I/O device (manufacturers part no. INS8154). This device can be added without any additional modification, and provides the kit user with a further 128 words of RAM and a set of 16 lines which can be utilised as logic inputs in any combination.
These 16 lines are designated Port A (8 lines) and Port B (8 lines) and are available at the edge connector as shown in Fig. 10.1.

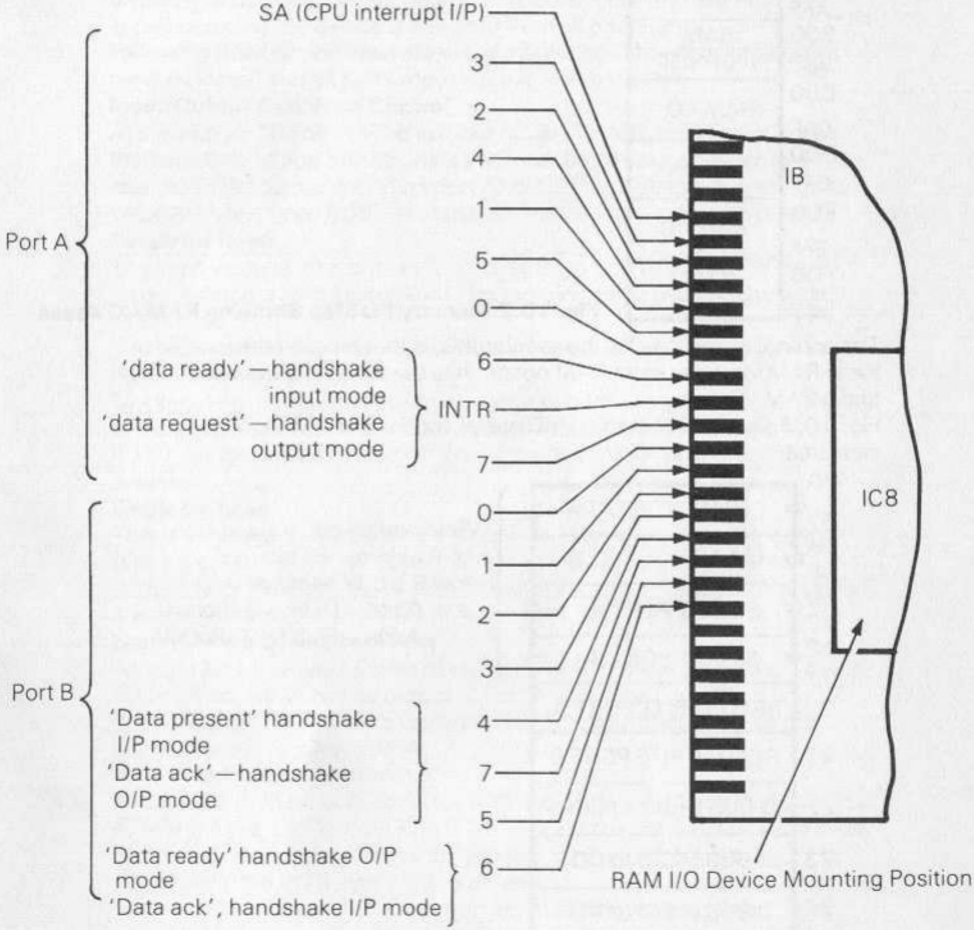
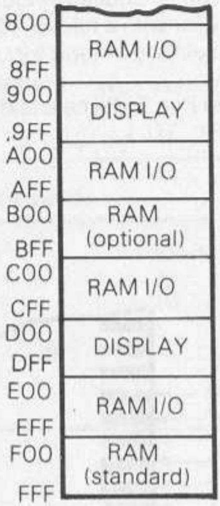


Fig. 10.1 RAM I/O Signal Lines

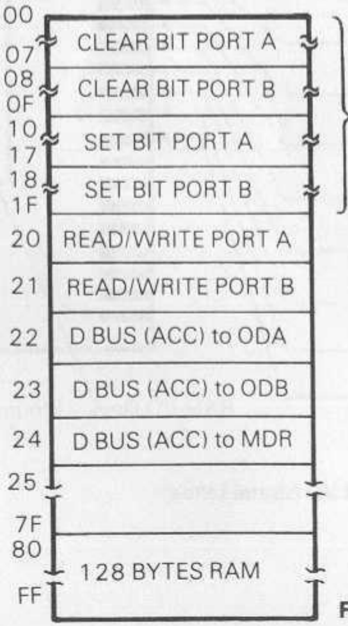
The RAM I/O can be regarded as two completely separate functional entities, one being the memory element and the other the input/output section. The only association between the two is that they share the same package and occupy adjacent areas in the memory I/O space. Fig. 10.2 shows the blocks in the memory map occupied by the RAM I/O, and it can be seen that the one piece of hardware is present in four separate blocks of memory.



Note: — Memory area is shown divided into 256 byte blocks. The lowest and highest location address is shown in hex' at left.

Fig. 10.2 Memory I/O Map Showing RAM I/O Areas

The primary advantage for the user, in this, is that programme located in basic RAM, or in the extra RAM option, has the same address relationship to the RAM I/O.
Fig. 10.3 shows how memory I/O space within the RAM I/O block is allocated.



Selected bit out of 8 determined by low 3 bits of address
e.g. Addr. = 0, bit = 0 (Port A)
Addr. = IF, bit = 7 (Port B)

Fig. 10.3 RAM I/O Locations and Related Functions

RAM Section
This is utilised in precisely the same manner as any other area of RAM.
Input/Output Section
The device incorporates circuitry which affords the user a great deal of flexibility in usage of the 16 input/output lines. Each line can be separately defined as either an input or an output under programme control. Each line can be independently either read as an input, or set to logic 'I' or 'O' as an output. These functions are determined by the address value employed.
A further group of usage modes permit handshake logic i.e. a 'data request', 'data ready', 'data receieved', signalling sequence to take place in conjunction with 8 bit parallel data transfers in or out through Port A.
Reset Control
This input from the RAM I/O is connected in parallel with the CPU power-on and manual reset. When reset is present all port lines are high impedance and the device is inhibited from all operations.
Following reset all port lines are set to input mode, handshake facilities are deselected and all port output latches are set to zero.
Input/Output Definition Control
At start-up all 16 lines will be in input mode. To convert a line or lines to the output condition a write operation must be performed by programme into the ODA (output definition port A) or ODB locations e.g. writing the value 80 (Hex.) into ODB will cause bit 7 port B to become an output.
Single Bit Read
The logic value at an input pin is transferred to the high order bit (bit 7) by performing a read instruction. The remaining bits in the accumulator become zero.
The required bit is selected by addressing the appropriate location (see Figs. 3 & 4).
By executing JP (Jump if Positive) instruction the programme can respond to the input signal i.e. the jump does not occur if the I/P is a logic 'i'.
If a bit designated as an output is read the current value of that O/P is detected.
Single Bit Load
This is achieved by addressing a write operation to a selected location (see Figs. 10.1 & 10.4). Note that it is not necessary to preset the accumulator to define the written bit value because it is determined by bit 4 of the address.
Eight Bit Parallel Read or Write
An eight bit value can be read from Port A or B to the accumulator, or the accumulator value can be output to Port A or B. See Figs. 10.3 & 10.4 for the appropriate address locations. Input/output lines must be pre-defined for the required mode.
Port A Handshake Operations
To achieve eight bit data transfers with accompanying handshake via Port A, two lines (6 and 7) from Port B are allocate special functions and must be pre-defined by programme as follows:- bit 7-input, bit 6-output.
Additionally the INTR signal line is utilised.
Three modes of handshake function are available to be selected under programme control. Fig. 10.4 shows values to be written into the three higher order bits of the Mode Definition Register (see Fig. 10.1 for location) for the various modes.

Bit Position & value in MDR

Mode	7	6	5
BASIC I/O (this condition selected by reset)	X	X	O
STROBED INPUT	X	O	I
STROBED OUTPUT	O	I	I
STROBED OUTPUT WITH TRI-STATE	I	I	I

Note:-
i) X = don't care
ii) Lower order bits are don't care also.

Fig. 10.4 Mode Definition Register (MDR) Values and Operation Modes

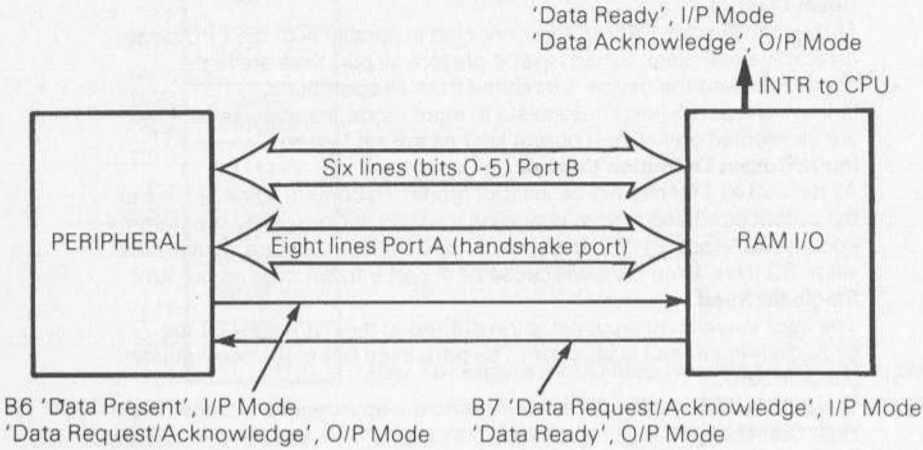


Fig. 10.5 Handshake Interconnections and Function

INTR Signal

In order to inform the CPU of the state of the data transfer in handshake mode the RAM I/O generates the INTR SIGNAL: This signal will usually be connected to the CPU interrupt input SA.

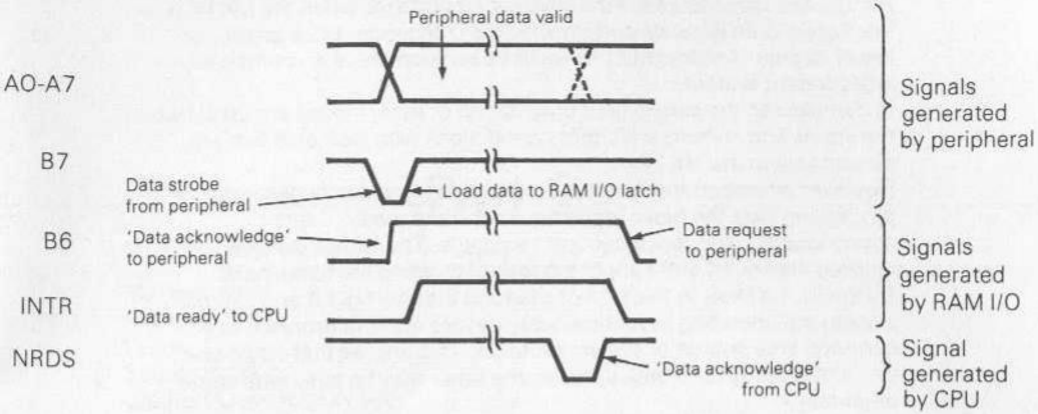
The INTR signal is activated by writing a logic 'I' into B7 and is inhibited by a logic 'O'. Note that although B7 must be defined as an input, in handshake mode the B7 output latch remains available to perform this special function.

Strobed Input Mode

A peripheral circuit applies a byte of information to Port A and a low pulse to B7. The pulse causes the data to be latched into the RAM I/O Port A register, and B6 is made high as a signal to the peripheral indicating that the latch is now occupied. At the same time INTR (if enabled) goes high indicating 'data ready' to the CPU.

The CPU responds with a byte read from Port A. The RAM I/O recognises this, and removes INTR and the 'buffer full' signal on B6, informing the peripheral that the latch is available for new data.

Fig. 10.6 Signal Timing Relationship – Handshake I/P Mode



Strobed Output Mode

The CPU performs a byte write to Port A, and the RAM I/O generates a 'data ready' signal by making B6 low. The peripheral responds to 'data ready' by accepting the Port A data, and acknowledges by making B7 low. When B7 goes low the RAM I/O makes INTR high (if enabled) informing the CPU that the data transaction is complete.

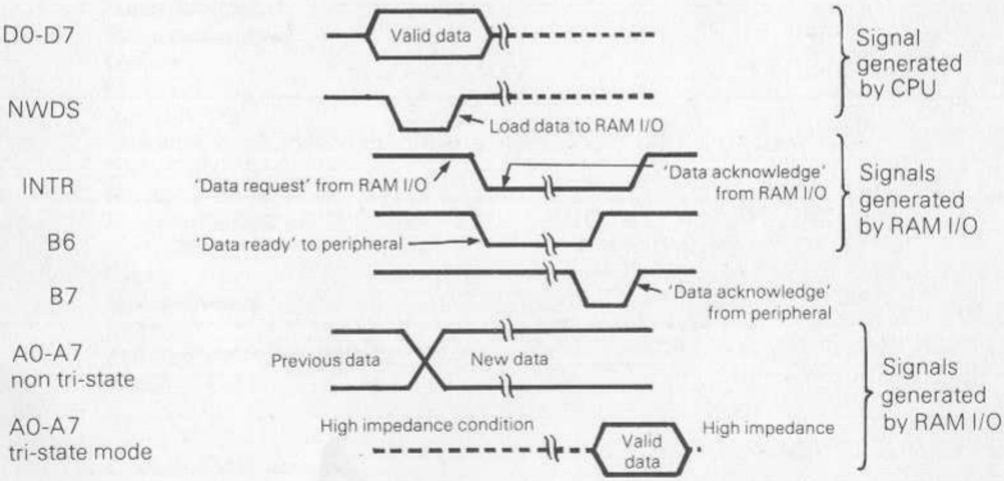


Fig. 10.7 Signal Timing Relationship – Handshake O/P Mode

Strobed Output with Tri-State Control

This mode employs the same signalling and data sequence as does Output Mode above. However the difference lies in that Port A will, in this mode, normally be in Tri-state condition (i.e. no load on peripheral bus), and will only apply data to the bus when demanded by the peripheral by a low acknowledge signal to B7.

Applications for Handshake Mode

Handshake facilities afford the greatest advantages when the MK14 is interfaced to an external system which is independent to a greater or lesser degree. Another MK14 would be an example of an completely independent system.

In comparison the simple read or write, bit or byte, modes are useful when the inputs and outputs are direct connections with elements that are subservient to the MK14.

However whenever the external system is independently generating and processing data the basic 'data request', 'data ready', 'data acknowledge', sequence becomes valuable. The RAM I/O first of all relieves the MK14 software of the task of creating the handshake.

Secondly it is likely in this kind of situation that the MK14 and external system are operating asynchronously i.e. are not synchronised to a common time source or system protocol. This implies that when one element is ready for a data transfer, the other may be busy with some other task.

Here the buffering ability of the Port A latch eases these time constraints by holding data transmitted by one element until the other is ready to receive.

Therefore, for example, if the CPU, in the position of a receiver, is unable, due to the requirements of the controlling software, in the worst case, to pay attention for 2 millisecs the transmitter would be allowed to send data once every millisecond.

Part 2

Devised and written by:
David Johnson—Davies
except programmes marked thus *

Monitor program listing

```
                         SCMPKB

SC/MP ASSEMBLER REV-C 02/06/76
SCMPKB   P005235A   7/14/76
                          TITLE SCMPKB, 'P005235A 7/14/76'
                 ;**************************************************
                 ;*
                 ;*
                 ;*                                  BOARD
                 ;*         PROM#         ADDRESS  COORDINATE   BOARD#
                 ;*
                 ;*    460305235-001      0000         5A       9804879
                 ;*
                 ;*
                 ;**************************************************


      0F00  RAM          =     0F00
      0D00  DISP         =     0D00

                 ;          SEGMENT ASSIGNMENTS

      0001  SA           =
      0002  SB           =

      0001  SA           =     1
      0002  SB           =     2
      0004  SC           =     4
      0008  SD           =     8
      0010  SE           =     16
      0020  SF           =     32
      0040  SG           =     64

                 ;          7  SEGMENT CONVERSION

      003F  N0           =     SA+SB+SC+SD+SE+SF
      0006  N1           =     SB+SC
      005B  N2           =     SA+SB+SD+SE+SG
      004F  N3           =     SA+SB+SC+SD+SG
      0066  N4           =     SB+SC+SF+SG
      006D  N5           =     SA+SC+SD+SF+SG
      007D  N6           =     SA+SC+SD+SE+SF+SG
      0007  N7           =     SA+SB+SC
      007F  N8           =     SA+SB+SC+SD+SE+SF+SG
      0067  N9           =     SA+SB+SC+SF+SG
      0077  NA           =     SA+SB+SC+SE+SF+SG
      007C  NB           =     SC+SD+SE+SF+SG
      0039  NC           =     SA+SD+SE+SF
      005E  ND           =     SB+SC+SD+SE+SG
      0079  NE           =     SA+SD+SE+SF+SG
      0071  NF           =     SA+SE+SF+SG
      0040  DASH         =     SG
      0079  KE           =     NE
      0050  KR       =         SE+SG
      005C  KO           =     SC+SD+SE+SG


                         .PAGE    'HARDWARE FOR KEYBOARD'


                 ;         FUNCTION   DATA   KYB FUNCTION

                 ;             0       080        0
                 ;             1       081        1
                 ;             2       082        2
```

```
                  ;         3       083       3
                  ;         4       084       4
                  ;         5       085       5
                  ;         6       086       6
                  ;         7       087       7
                  ;       8      040      8
                  ;       9      041      9
                  ;       A      010      +
                  ;       B      011      —
                  ;       C      012      MUL
                  ;       D      013      DIV
                  ;       E      016      SQUARE
                  ;       F      017      SQRT
                  ;       GO     022      %
                  ;       MEM    023      =
                  ;       ABORT  024      CE/C
                  ;       TERM   027

                  ;       RAM POINTERS USED BY KITBUG, P3 IS SAVED ELSEWHERE


      0FF9  P1H           =      0FF9
      0FFA  P1L           =      0FFA
      0FFB  P2H           =      0FFB
      0FFC  P2L           =      0FFC
      0FFD  A             =      0FFD
      0FFE  E             =      0FFE
      0FFF  S             =      0FFF
                  ;       COMMANDS

                  ;ABORT:
                  ;       THIS ABORTS THE PRESENT OPERATION. DISPLAYS—.

                  ;MEM:
                  ;       ALLOWS USER TO READ/MODIFY MEMORY.
                  ;       ADDRESS IS ENTERED UNTIL TERM THEN DATA IS ENTERED.
                  ;       TO WRITE DATA IN MEMORY TERM IS PUSHED.
                  ;       DATA IS READ TO CHECK IF IT GOT WRITTEN IN RAM.

                  ;GO:
                  ;       ADDRESS IS ENTERED UNTIL TERM.
                  ;       THE REGISTERS ARE LOADED FROM RAM AND PROGRAM
                  ;       IS TRANSFERRED USING XPPC P3.
                  ;       TO GET BACK DO A XPPC P3.


                          .PAGE 'INITIALIZE'
0000  08                  NOP
0001              INIT:
0001  901D                JMP      START

                  ;       DEBUG EXIT
                  ;       RESTORE ENVIRONMENT

0003              GOOUT:
0003  C20E                LD      ADH(2)     ;GET GO ADDRESS.
0005  37                  XPAH    3
0006  C20C                LD      ADL(2)
0008  33                  XPAL    3
0009  C7FF                LD      @-1(3)     ;FIX GO ADDRESS.
000B  C0F2                LD      E          ;RESTORE REGISTERS.
000D  01                  XAE
000E  C0EB                LD      P1L
0010  31                  XPAL    1
0011  C0E7                LD      P1H
0013  35                  XPAH    1
0014  C0E7                LD      P2L
0016  32                  XPAL    2
0017  C0E3                LD      P2H
0019  36                  XPAH    2
001A  C0E4                LD      S
```

```
     001C 07              CAS
     001D C0DF            LD    A
     001F 3F              XPPC  3
                                          ;TO BET BACK.
                   ;      ENTRY POINT FOR DEBUG
     0020         START:
     0020 C8DC            ST    A         ;SAVE STATUS.
     0022 40              LDE
     0023 C8DA            ST    E
     0025 06              CSA
     0026 C8D8            ST    S
     0028 35              XPAH  1
     0029 C8CF            ST    P1H
     002B 31              XPAL  1
     002C C8CD            ST    P1L
     002E C40F            LDI   H(RAM)    ;SET P2 TO POINT TO RAM.
     0030 36              XPAH  2
     0031 C8C9            ST    P2H
     0033 C400            LDI   L(RAM)
     0035 32              XPAL  2
     0036 C8C5            ST    P2L
     0038 C701            LD    @1(3)     ;BUMP P3 FOR RETURN.
     003A 33              XPAL  3         ;SAVEp3.
     003B CA0C            ST    ADL(2)
     003D 37              XPAH  3
     003E CA0E            ST    ADH(2)
                          .PAGE
                   ;      ABORT SEQUENCE
     0040         ABORT:
     0040 C400            LDI   0
     0042 CA02            ST    D3(2)
     0044 CA03            ST    D4(2)
     0046 CA08            ST    D9(2)
     0048 C440            LDI   DASH      ;SET SEGMENTS TO—.
     004A CA00            ST    DL(2)
     004C CA01            ST    DH(2)
     004E CA04            ST    ADDLL(2)
     0050 CA05            ST    ADLH(2)
     0052 CA06            ST    ADHL(2)
     0054 CA07            ST    ADHH(2)
     0056         WAIT:
     0056 C401            JS    3,KYBD    ;DISPLAY AND READ KEYBOAF
     0058 37C4
     005A 8433
     005C 3F
     005D 9002            JMP   WCK       ;COMMAND RETURN.
     005F 90DF            JMP   ABORT     ;RETURN FOR NUMBER.
     0061         WCK:
     0061 E407            XRI   07        ;CHECK IF MEM.
     0063 9856            JZ    MEM
     0065 E401            XRI   01        ;CHECK IF GO.
     0067 9CD7            JNZ   ABORT
                          .PAGE 'GO TO'
                   ;      GO WAS PUSHED
                   ;      GO TO USER PROGRAM
     0069         GO:
     0069 C4FF            LDI   —1        ;SET FIRST FLAG.
     006B CA0F            ST    DDTA(2)
     006D C440            LDI   DASH      ;SET DATA TO DASH.
     006F CA00            ST    DL(2)
     0071 CA01            ST    DH(2)
     0073         GOL:
     0073 C459            LDI   L(DISPA)-1 ;FIX ADDRESS SEG.
```

```
0075 33              XPAL  3
0076 3F              XPPC  3             ;DO DISPLAY AND KEYBRD.
0077 9006            JMP   GOCK          ;COMMAND RETURN.
0079 C41A            LDI   L(ADR)-1      ;SET ADDRESS.
007B 33              XPAL  3
007C 3F              XPPC  3
007D 90F4            JMP   GOL           ;NOT DONE.
007F           GOCK:
007F E403            XRI   03            ;CHECK FOR TERM.
0081 9880            JZ    GOOUT         ;ERROR IF NO TERM.
               ;     INCORRECT SEQUENCE
               ;     DISPLAY ERROR WAIT FOR NEW INPUT
0083           ERROR:
0083 C479            LDI   KE            ;FILL WITH ERROR.
0085 CA07            ST    ADHH(2)
0087 C450            LDI   KR
0089 CA06            ST    ADHL(2)
008B CA05            ST    ADLH(2)
008D CA03            ST    D4(2)
008F C45C            LDI   KO
0091 CA04            ST    ADLL(2)
0093 C400            LDI   0
0095 CA02            ST    D3(2)
0097 CA01            ST    DH(2)
0099 CA00            ST    DL(2)
009B 90B9            JMP   WAIT
                     .PAGE 'MEMORY TRANSACTIONS'
009D           DTACK:
009D C211            LD    NEXT(2)       ;CHECK IF DATA FIELD.
009F 9C36            JNZ   DATA          ;ADDRESS DONE.
00A1           MEMDN:
00A1 C20E            LD    ADH(2)        ;PUT WORD IN MEM.
00A3 35              XPAH  1
00A4 C20C            LD    ADL(2)
00A6 31              XPAL  1
00A7 C20D            LD    WORD(2)
00A9 C900            ST    (1)
00AB 900E            JMP   MEM
00AD           MEMCK:
00AD E406            XRI   06            ;CHECK FOR GO.
00AF 98D2            JZ    ERROR         ;CAN NOT GO NOW.
00B1 E405            XRI   05            ;CHECK FOR TERM.
00B3 98E8            JZ    DTACK         ;CHECK IF DONE.
00B5 AA0C            ILD   ADL(2)        ;UPDATE ADDRESS LOW.
00B7 9C02            JNZ   MEM           ;CHECK IF UPDATE HI.
00B9 AA0E            ILD   ADH(2)
               ;     MEM KEY PUSHED
00BB           MEM:
00BB C4FF            LDI   -1            ;SET FIRST FLAG.
00BD CA11            ST    NEXT(2)       ;SET FLAG FOR ADDRESS NOW.
00BF CA0F            ST    DDTA(2)
00C1           MEML:
00C1 C20E            LD    ADH(2)
00C3 35              XPAH  1             ;SET P1 FOR MEM ADDRESS.
00C4 C20C            LD    ADL(2)
00C6 31              XPAL  1
00C7 C100            LD    (1)
00C9 CA0D            ST    WORD(2)       ;SAVE MEM DATA.
00CB C43F            LDI   L(DISPD)-1    ;FIX DATA SEG.
00CD 33              XPAL  3
00CE 3F              XPPC  3             ;GO TO DISPD SET SEG FOR DATA.
```

```
00CF 90DC           JMP   MEMCK       ;COMMAND RETURN.
00D1 C41A           LDI   L(ADR)-1    ;MAKE ADDRESS.
00D3 33             XPAL  3
00D4 3F             XPPC  3
00D5 90EA           JMP   MEML        ;GET NEXT CHAR.
00D7           DATA:
00D7 C4FF           LDI   -1          ;SET FIRST FLAG.
00D9 CA0F           ST    DDTA(2)
00DB C20E           LD    ADH(2)      ;SET P1 TO MEMORY ADDRESS.
00DD 35             XPAH  1           275
00DE C20C           LD    ADL(2)
00E0 31             XPAL  1
00E1 C100           LD    (1)         ;READ DATA WORD.
00E3 CA0D           ST    WORD(2)     ;SAVE FOR DISPLAY.

                    .PAGE
00EE5          DATAL:
00E5 C43F           LDI   L(DISPD)-1  ;FIX DATA SEG.
00E7 33             XPAL  3
00E8 3F             XPPC  3           ;FIX DATA SEG-GO TO DISPD.
00E9 90C2           JMP   MEMCK       ;CHAR RETURN.
00EB C404           LDI   4           ;SET COUNTER FOR NUMBER OF SHIFTS.
00ED CA09           ST    CNT(2)
00EF AA0F           ILD   DDTA(2)     ;CHECK IF FIRST.
00F1 9C06           JNZ   DNFST
00F3 C400           LDI   0           ;ZERO WORD IF FIRST.
00F5 CaOD           ST    WORD(2)
00F) CA11           ST    NEXT(2)     ;SET FLAG FOR ADDRESS DONE.
00F9           DNFST:
00F9 02             CCL
00FA C20D           LD    WORD(2)     ;SHIFT LEFT.
00FC F20D           ADD   WORD(2)
00FE CA0D           ST    WORD(2)
0100 BA09           DLD   CNT(2)      ;CHECK FOR 4 SHIFTS.
0102 9CF5           JNZ   DNFST
0104 C20D           LD    WORD(2)     ;ADD NEW DATA.
0104 C296
0104 C206           LD    WORD(2)     ;ADD NEW DATA.
0106 58             ORE
0107 660D CAOD      ST    WORD(2)
0109 90DA           JMP   DATAL
0109 96DA           JMP   DATAL

                    .PAGE 'HEXNUMBBER TO SEGMENT TABLE'

               ;    'HEX NUMBER TO  SEVEN SEGMENT TABLE'


010B           CROM:
010B 3F             .BYTE N0
010C 06             .BYTE N1
010D 5B             .BYTE N2
010E 4F             .BYTE N3
010F 66             .BYTE N4
0110 6D             .BYTE N5
0111 7D             .BYTE N6
0112

0111 7A             .BYTE N6
0112 07             .BYTE N7
0113 7F             .BYTE N8
0114 67             .BYTE N9
0115 77             .BYTE NA
0116 7C             .BYTE NB
0117 39             .BYTE NC
0118 5E             .BYTE ND
0119 79             .BYTE NE
011A 71             .BYTE NF

                    .PAGE 'MAKE 4 DIGIT ADDRESS'
011B           ADR:
```

```
                    ;         SHIFT ADDRESS LEFT ONE DIGIT THEN
                    ;

                    ;         SHIFT ADDRESS LEFT ONE DIGIT THEN
                    ;         ADD NEW LOW HEX DIGIT.
                    ;         HEX DIGIT IN E REGISTER.
                    ;         P2 POINTS TO RAM.

011B  C404                    LDI   4          ;SET NUMBER OF SHIFTS.
011D  CA09                    ST    CNT(2)
011F  AA0F                    ILD   DDTA(2)    ;CHECK IF FIRST.
0121  9C06                    JNZ   NOTFST     ;JMP IF NO.
0123  C400                    LDI   0          ;ZERO ADDRESS.
0125  CA0E                    ST    ADH(2)
0127  CA0C                    ST    ADL(2)
0129                NOTFST:
0129  02                      CCL              ;CLEAR LINK.
012A  C20C                    LD    ADL(2)     ;SHIFT ADDRESS LEFT 4 TIMES.
012C  F20C                    ADD   ADL(2)
012E  CA0C                    ST    ADL(2)     ;SAVE IT.
0130  C20E                    LD    ADH(2)     ;NOW SHIFT HIGH.
0132  F20E                    ADD   ADH(2)
0134  CA0E                    ST    ADH(2)
0136  BA09                    DLD   CNT(2)     ;CHECK IF SHIFTED 4 TIMES.
0138  9CEF                    JNZ   NOTFST     ;JMP IF NOT DONE.
013A  C20C                    LD    ADL(2)     ;NOW ADD NEW NUMBER.
013C  58                      ORE
013D  CA0C                    ST    ADL(2)     ;NUMBER IS NOW UP DATED.
013F  3F                      XPPC  3

                              .PAGE 'DATA TO SEGMENTS'

                    ;         CONVERT HEX DATA TO SEGMENTS.
                    ;         P2 POINTS TO RAM.
                    ;         DROPS THRU TO HEX ADDRESS CONVERSION.

0140                DISPD:
0140  C401                    LDI   H(CROM)    ;SET ADDRESS OF TABLE.
0142  35                      XPAH  1
0143  C40B                    LDI   L(CROM)
0145  31                      XPAL  1
0146  C20D                      ld  word62)    ;GET MEMORY WORD.
0148  D40F                    ANI   0F
014A  01                      XAE
014B  C180                    LD    -128(1)    ;GET SEGMENT DISP.
014D  CA00                    ST    DL(2)      ;SAVE AT DATA LOW.
014F  C20D                    LD    WORD(2)    ;FIX HI.
0151  1C                      SR               ;SHIFT HI TO LOW.
0152  1C                      SR
0153  1C                      SR
0154  1C                      SR
0155  01                      XAE
0156  C180                    LD    -128(1)    ;GET SEGMENTS.
0158  CA01                    ST    DH(2)      ;SAVE IN DATA HI.

                    .PAGE     ADDRESS TO SEGMENTS

                    ;         CONVERT HEX ADDRESS TO SEGMENTS.
                    ;         P2 POINTS TO RAM.
```

```
                   ;         DROPS THRU TO KEYBOARD AND DISPLAY.

015A               DISPA:
015A 03                      SCL
015B C401                    LDI   H(CROM)    ;SET ADDRESS OF TABLE.
015D 35                      XPAH  1
015E C40B                    LDI   L(CROM)
0160 31                      XPAL  1
0161               LOOPD:
0161 C20C                    LD    ADL(2)     ;GET ADDRESS.
0163 D40F                    ANI   OF
0165 01                      XAE
0166 C180                    LD               ;GET SEGMENTS.
0168 CA04                    ST    ADLL(2)    ;SAVE SEG OF ADR LL.
016A C20C                    LD    ADL(2)
016C 1C                      SR               ;SHIFT HI DIGIT TO LOW.
016D ...c                    SR
016E 1C                      SR
016F 1                   SR
0170 01                      XAE
0171 C180                    LD    -128(1)    ;GET SEGMENTS.
0173 CA05                    ST    ADLH(2)
0175 06                      CSA              ;CHECK IF DONE.
0176 D480                    ANI   080
0178 9809                    JZ    DONE
017A 02                      CCL              ;CLEAR FLAG.
017B C400                    LDI   0
017D CA03                    ST    D4(2)      ;ZERO DIGIT 4.
017F C602                    LD    @2(2)      ;FIX P2 FOR NEXT LOOP.
0181 90DE                    JMP   LOOPD
0183               DONE:
0183 C6FE                    LD    @-2(2)     ;FIX P2.


                   .PAGE 'DISPLAY AND KEYBOARD INPUT'

                   ;         CALL  XPPC  3

                   ;         JMP COMMAND IN A GO = 6,MEM = 7,TERM = 3
                   ;                IN E GO = 22,MEM = 23,TERM = 27.
                   ;         NUMBER RETURN HEX NUMBER IN E REG.

                   ;         ABORT KEY GOES TO ABORT.
                   ;         ALL REGISTERS ARE USED.

                   ;         P2 MUST POINT TO RAM. ADDRESS MUST BE XXXO.

                   ;         TO RE-EXECUTE ROUTINE DO XPPC P3.


0185               KYBD:
0185 C400                    LDI   0          ;ZERO CHAR.
0187 CA0B                    ST    CHAR(2)
0189 C40D                    LDI   H(DISP)    ;SET DISPLAY ADDRESS.
018B 35                      XPAH  1
018C               OFF:
018C C4FF                    LDI   -1         ;SET ROW/DIGIT ADDRESS.
018E CA10                    ST    ROW(2)     ;SAVE ROW COUNTER.
0190 C40A                    LDI   10         ;SET ROW COUNT.
0192 CA09                    ST    CNT(2)
0194 C400                    LDI   0
0196 CA0A                    ST    PUSHED(2)  ;ZERO KEYBOARD INPUT.
0198 31                      XPAL  1          ;SET DISP ADDRESS LOW.
0199               LOOP:
0199 AA10                    ILD   ROW(2)     ;UP DATE ROW ADDRESS.
019B 01                      XAE
019C C280                    LD    -128(2)    ;GET SEGMENT.
019E C980                    ST    -128(1)    ;SEND IT.
01A0 8F00                    DLY   0          ;DELAY FOR DISPLAY.
```

```
01A2  C180           LD    -128(1)     ;GET KEYBOARD INPUT.
01A4  E4FF           XRI   OFF         ;CHECK IF PUSHED.
01A6  9C4C           JNZ   KEY         ;JUMP IF PUSHED.
01A8         BACK:
01A8  BA09           DLD   CNT(2)      ;CHECK IF DONE.
01AA  9CED           JNZ   LOOP        ;NO IF JUMP.
01AC  C20A           LD    PUSHED(2)   ;CHECK IF KEY.
01AE  980A           JZ    CKMORE
01B0  C20B           LD    CHAR(2)     ;WAS THERE A CHAR?
01B2  9CD8           JNZ   OFF         ;YES WAIT FOR RELEASE.
01B4  C20A           LD    PUSHED(2)   ;NO SET CHAR.
0...B6 CA0B          ST    CHAR(2)
01B8  90D2           JMP   OFF
01BA         CKMORE:
01BA  C20B           LD    CHAR(2)     ;CHECK IF THERE WAS A CHAR.
01BC  98CE           JZ    OFF         ;NO KEEP LOOKING.

                     .PAGE

             ;       COMMAND KEY PROCESSING

01BE         COMMAND:
01BE  01             XAE               ;SAVE CHAR.
01BF  40             LDE               ;GET CHAR.
01C0  D420           ANI   020         ;CHECK FOR COMMAND.
01C2  9C28           JNZ   CMND        ;JUMP IF COMMAND.
01C4  C480           LDI   080         ;FIND NUMBER.
01C6  50             ANE
01C7  9C1B           JNZ   LT7         ;0 TO 7.
01C9  C440           LDI   040
01CB  50             ANE
01CC  9C19           JNZ   N89         ;8 OR 9.
01CE  C40F           LDI   OF
01D0  50             ANE
01D1  F407           ADI   7           ;MAKE OFF SET TO TABLE.
01D3  01             XAE               ;PUT OFF SET AWAY.
01D4  C080           LD    -128(0)     ;GET NUMBER.
01D6         KEYRTN:
01D6  01             XAE               ;SAVE IN E.
01D7  C702           LD    @2(3)       ;FIX RETURN.
01D9  3F             XPPC  3           ;RETURN.
01DA  90A9           JMP   KYBD        ;ALLOWS XPPC P3 TO RETURN.

01DC  0A0B           .BYTE 0A, 0B, 0C, 0D, 0, 0E, 0F
01DE  0C0D
01E0  0000
01E2  0E0F
01E4         LT7:
01E4  60             XRE               ;KEEP LOW DIGIT.
01E5  90EF           JMP   KEYRTN
01E7         N89:
01E7  60             XRE                     ;GET LOW.
01E8  F408           ADI      08             ;MAKE DIGIT 8 OR 9.
01EA  90EA           JMP      KEYRTN

                     .PAGE
01EC         CMND:
01EC  60             XRE
01ED  E404           XRI      04             ;CHECK IF ABORT.
01EF  9808           JZ       ABRT           ;ABORT.
01F1  3F             XPPC     3              ;IN E 23 = MEM, 22 = GO, 27 = TERM
                                             ;IN A 7 = MEM, 6 = GO, 3 = TERM.
01F2  9091           JMP      KYBD           ;ALLOWS JUST A XPPC P3 TO
                                             ;RETURN.

01F4         KEY:
01F4  58             ORE                     ;MAKE CHAR.
01F5  CA0A           ST       PUSHED(2)      ;SAVE CHAR.
01F7  90AF           JMP      BACK

01F9         ABRT:
```

```
01F9 C400          LDI      H(ABORT)
01FB 37            XPAH     3
01FC C43F          LDI      L(ABORT)-1
01FE 33            XPAL     3
01FF 3F            XPPC     3                 ;GO TO ABORT

                   .PAGE    'RAM      SEØFF-


     0000 DL       =        0                 ;SEGMENT FOR DIGIT 1
     0001 DH       =        1                 ;SEGMENT FOR DIGIT 2
     0002 D3       =        2                 ;SEGMENT FOR DIGIT 3
     0003 D4       =        3                 ;SEGMENT FOR DIGIT 4
     0004 ADLL     =        4                 ;SEGMENT FOR DIGIT 5
     0005 ADLH     =        5                 ;SEGMENT FOR DIGIT 6
     0006 ADHL     =        6                 ;SEGMENT FOR DIGIT 7
     0007 ADHH     =        7                 ;SEGMENT FOR DIGIT 8
     0008 D9       =        8                 ;SEGMENT FOR DIGIT 9
     0009 CNT      =        9                 ;COUNTER.
     000A PUSHED   =        10                KEY PUSHED.
     000B CHAR     11

     000B CHAR     =        11                ;CHAR READ.
     000C ADL      =        12                ;MEMORY ADDRESS LOW.
     000D WORD     =        13                ;MEMORY WORD.
     000E ADH      =        14                ;MEMORY ADDRESS HI.
     000F =        =        15                ;FIRST FLAG.
     0010 ROW      =        16                ;ROW COUNTER.
     0011 NEXT     =        17                ;FLAG FOR NOW DATA.


     0000                   .END
```

******* 0 ERRORS IN ASSEMBLY ******

A	ABORT	ABRT	ADH	ADHH	ADHL	ADL	ADLH	ADLL	ADR
0FFD	0040	01F9	000E	0007	0006	000C	0005	0004	011B
BACK	CHAR	CKMORE	CMND	CNT	COMMAN	CROM	D3	D4	D9
01A8	000B	01BA	01EC	0009	01BE	010B	0002	0003	0008
DASH	DATA	DATAL	DDTA	DH	DISP	DISPA	DISPD	DL	DNFST
0040	00D7	00E5	000F	0001	0D00	015A	0140	0000	00F9
DONE	DTACK	E	ERROR	GO	GOCK	GOL	GOOUT	INIT	KE
0183	009D	0FFE	0083	0069	007F	0073	0003	0001	0079
KEY	KEYRTN	KO	KR	KYBD	LOOP	LOOPD	LT7	MEM	MEMCK
01F4	01D6	005C	0050	0185	0199	0161	01E4	00BB	00AD
MEMDN	MEML	N0	N1	N2	N3	N4	N5	N6	N7
00A1	00C1	003F	0006	005B	004F	0066	006D	007D	0007
N8	N89	N9	NA	NB	NC	NC	NE	NEXT	NF
007F	01E7	0067	0077	007C	0039	005E	0079	0011	0071
NOTFST	OFF	P1H	P1L	P2H	P2L	PUSHED	RAM	ROW	S
0129	018C	0FF9	0FFA	0FFB	0FFC	000A	0F00	0010	0FFF
SA	SB	SC	SD	SE	SF	SG	START	WAIT	WCK
0001	0002	0004	0008	0010	0020	0040	0020	0056	0061
WORD									
000D									

A799 08AB

Mathematical

The mathematical subroutines all take their arguments relative to the pointer register P2. Pointer P3 is the subroutine calling register. All of these routines may be repeated without reloading P3 after the first call.

'Multiply' gives the 16-bit unsigned product of two 8-bit unsigned numbers.

e.g. A = X'FF (255)
B = X'FF (255)
RESULT = X'FE01 (65025).

'Divide' gives the 16-bit unsigned quotient and 8-bit remainder of a 16-bit unsigned dividend divided by an 8-bit unsigned divisor.

e.g. DIVISOR = X'05 (5)
DIVISOR = X'5768 (22376)
QUOTIENT = X'117B (4475)
REMAINDER = X'01 (1).

'Square Root' gives the 8-bit integer part of the square root of a 16-bit unsigned number. It uses the relation:

$(n+1)^2 - n^2 = 2n+1$,

and subtracts as many successive values of $2n + 1$ as possible from the number, thus obtaining n.

e.g. NUMBER = X'D5F6 (54774)
ROOT = X'EA (234).

'Greatest Common Divisor' uses Euclid's algorithm to find the GCD of two 16-bit unsigned numbers; i.e. the largest number which will exactly divide them both. If they are coprime the result is 1.

e.g. A = X'FFCE (65486 = 478 × 137)
B = X'59C5 (23701 = 173 × 137)
GCD = X'89 (137).

Multiply

```
                ; Multiplies two unsigned 8-bit numbers
                ; (Relocatable)
                ;
                ; Stack usage:
                ;           REL:   ENTRY:   USE:        RETURN:
                ;           -1              Temp
                ;(P2)->      0     A        A           A
                ;            1     B        B           B
                ;            2              Result (H)  Result (H)
                ;            3              Result (L)  Result (L)
                ;
0000    A          =        0
0001    B          =        1
FFFF    Temp       =        -1
0002    RH         =        2
0003    RL         =        3
                ;
```

```
0000                    .=0F50
0F50   C408   Mult:     LDI    8
0F52   CAFF             ST     Temp (2)
0F54   C400             LDI    0
0F56   CA02             ST     RH(2)
0F58   CA03             ST     RL(2)
0F5A   C201   Nbit:     LD     B(2)
0F5C   02               CCL
0F5D   1E               RR
0F5E   CA01             ST     B(2)
0F60   9413             JP     Clear
0F62   C202             LD     RH(2)
0F64   F200             ADD    A(2)
0F66   IF     Shift:    RRL
0F67   CA02             ST     RH(2)
0F69   C203             LD     RL(2)
0F6B   IF               RRL
0F6C   CA03             ST     RL(2)
0F6E   BAFF             DLD    Temp(2)
0F70   9CE8             JNZ    Nbit
0F72   3F               XPPC   3
0F73   90DB             JMP    Mult
0F75   C202   Clear:    LD     RH(2)
0F77   90ED             JMP    Shift
              ;
       0000             .END
```

Divide

```
              ; Divides an unsigned 16-bit number by
              ; an unsigned 8-bit number giving
              ; 16-bit quotient and 8-bit remainder.
              ; (Relocatable)
              ;
              ; Stack usage:
              ;         REL:    ENTRY:        USE:          RETURN:
              ;         -1                    Quotient(l)
              ;(P2)->    0      Divisor                     Quotient(H)
              ;         +1      Dividend(H)                 Quotient(L)
              ;         +2      Dividend(L)                 Remainder
              ;
       FFFF   Quot      =       -1
       0000   DSOR      =       0
       0001   DNDH      =       1
       0002   DNDL      =       2
              ;
0000                    .=0F80
0F80   C200   Div:      LD      DSOR(2)
0F82   01               XAE
0F83   C400             LDI     0
0F85   CA00             ST      DSOR(2)  ;Now Quotient(H)
```

```
0F87   CAFF             ST     Quot(2)  ;Quotient(L)
0F89   C201    Subh:    LD     DNDH(2)
0F8B   03               SCL
0F8C   78               CAE
0F8D   CA01             ST     DNDH(2)
0F8F   1D               SRL
0F90   9404             JP     Stoph
0F92   AA00             ILD    DSOR(2)
0F94   90F3             JMP    Subh
0F96   C201    Stoph:   LD     DNDH(2)
0F98   70               ADE             ;Carry is clear
0F99   CA01             ST     DNDH(2)  ;Undo damage
0F9B   C202    Subl:    LD     DNDL(2)
0F9D   03               CCL
0F9E   78               CAE
0FA0   CA02             ST     DNDL(2)
0FA2   C201             LD     DNDH(2)
0FA4   FC00             CAI    0
0FA6   CA01             ST     DNDH(2)
0FA8   1D               SRL
0FA9   9404             JP     Stopl
0FAB   AAFF             ILD    Quot (2)
0FAD   90ED             JMP    Subl
0FAF   C202    Stopl:   LD     DNDL(2)
0FB1   70               ADE
0FB2   CA02             ST     DNDL(2)  ;Remainder
0FB4   C2FF             LD     Quot(2)
0FB6   CA01             ST     DNDH(2)
0FB8   3F               XPPC   3        ;Return
0FB9   90C6             JMP    Div

       0000             .END
```

Square Root

```
                ; Gives square root of 16-bit unsigned number
                ; Integer part only. (Relocatable).
                ;
                ; Stack usage:
                ;        REL:     ENTRY:      USE:    RETURN:
                ;        -1                   Temp
                ;(P2)->   0       Number(H)           Root(H)
                ;        +1       Number(L)           Root(L)
                ;
       0000     HI       =        0
       0001     LO       =        1
       FFFF     Temp     =        -1
                ;
       0000              .=0F20
0F20   C400     SQRT:    LDI      X'00
0F22   CAFF              ST       Temp(2)
```

```
OF24   03     Loop:  SCL
OF25   BAFF          DLD    Temp(2)
OF27   F2FF          ADD    Temp(2)
OF29   01            XAE
OF2A   C4FE          LDI    X'FE
OF2C   F400          ADI    X'00
OF2E   01            XAE
OF2F   F201          ADD    LO(2)
OF31   CA01          ST     LO(2)
OF33   40            LDE
OF34   F200          ADD    HI(2)
OF36   CA00          ST     HI(2)
OF38   ID            SRL
OF39   9402          JP     EXIT
OF3B   90E7          JMP    LOOP
OF3D   C400   Exit:  LDI    X'00
OF3F   CA00          ST     HI(2)
OF41   FAFF          CAD    Temp(2)
OF43   CA01          ST     LO(2)
OF45   3F            XPPC   3          ;Return
OF46   90D8          JMP    SQRT       ;For Repeat
              ;
OF48                 .=OFFB
              ;
OFFB   OF80          .DBYTE OF80       ;P2-> Number

       0000          .END
```

Greatest Common Divisor

```
              ; Finds Greatest Common Divisor of two
              ; 16-bit unsigned numbers
              ; uses Euclid's Algorithm. (Relocatable).
              ;
              ; Stack usage:
              ;        REL:   ENTRY:   USE:    RETURN:
              ;(P2)->  0      A(H)     A(H)    0
              ;        1      A(L)     A(L)    0
              ;        2      B(H)     B(H)    GCD(H)
              ;        3      B(L)     B(L)    GCD(L)
              ;
       0000   AH     =      0
       0001   AL     =      1
       0002   BH     =      2
       0003   BL     =      3
              ;
0000                 .=OF20
OF20   03     GCD:   SCL
OF21   C203          LD     BL(2)
OF23   FA01          CAD    AL(2)
OF25   CA03          ST     BL(2)
OF27   01            XAE
```

```
0F28  C202          LD    BH(2)
0F2A  FA00          CAD   AH(2)
0F2C  CA02          ST    BH(2)
0F2E  1D            SRL                 ; Put carry in top bit
0F2F  9402          JP    Swap
0F31  90ED          JMP   GCD           ;Subtract again
0F33  02     Swap:  CCL
0F34  C201          LD    AL(2)
0F36  01            XAE
0F37  70            ADE
0F38  CA01          ST    AL(2)
0F3A  40            LDE
0F3B  CA03          ST    BL(2)
0F3D  C200          LD    AH(2)
0F3F  01            XAE
0F40  C202          LD    BH(2)
0F42  70            ADE
0F43  CA00          ST    AH(2)
0F45  01            XAE
0F46  CA02          ST    BH(2)
0F48  40            LDE                 ;Get new AH(2)
0F49  DA01          OR    AL(2)         ;OR with new AL(2)
0F4B  9CD3          JNZ   GCD           ;Not finished yet
0F4D  3F            XPPC  3             ;Return
0F4E  90D0          JMP   GCD           ;For repeat run

      0000          .END
```

Electronic

'Pulse Delay' uses a block of memory locations as a long shift-register, shifting bits in at the serial input SIN and out from the serial output SOUT. By varying the delay constants the input waveform can be delayed by up to several seconds, though for a fixed block of memory the resolution of the delay chain obviously decreases with increased delay.

With the program as shown the shift-register uses the 128 locations 0F80 to 0FFF, thus providing a delay of 1024 bits.

The 'Digital Alarm Clock' gives a continuously changing display of the time in hours, minutes and seconds. In addition, when the alarm time stored in memory tallies with the actual time the flag outputs are taken high. The time can be set in locations 0F16, 0F17, and 0F18, and the alarm time is stored in locations 0F12, 0F13, and 0F14.

The program depends for its timing on the execution time of the main loop of the program, which is executed 80 times a second, so this is padded out to exactly 1/80th of a second with a delay instruction. The delay constants at 0F7F and 0F81 should be adjusted to give the correct timing.

'Random Noise' generates a pseudo-random sequence of 2^{15}-1 or 65535 bits at the flag outputs. If one flag output is connected to an amplifier the sequence sounds like random noise. Alternatively, by converting the program to a subroutine to return one bit it could be used to generate random coin-tosses for games and simulations. Note that the locations 0F1E and 0F1F must not contain 00 for the sequence to start.

Pulse Delay

```
                        ; Pulse delayed by 1024 bit-times.
                        ; (Relocatable). Uses serial in/out.
                        ;
0000                              .=0F1F
0F1F                    Bits:     .=.+1               ;bit counter
                        ;
0F20    C40F            Enter:    LDI     H(Scrat)
0F22    35                        XPAH    1
0F23    C480                      LDIL    (Scrat)
0F25    31              Next:     XPAL    1
0F26    C408                      LDI     8
0F28    C8F6                      ST      Bits
0F2A    C100                      LD      (1)         ;Get old byte
0F2C    01                        XAE                 ;Exchange
0F2D    CD01                      ST      @+1(1)      ;Put back new byte
0F2F    19              Output:   SIO                 ;Serial I/O
0F30    C400                      LDI     TC1
0F32    8F04                      DLY     TC2         ;Delay bits
0F34    B8 EA                     DLD     Bits
0F36    9CF7                      JNZ     Output
0F38    31                        XPAL    1           ;P1=0D00 Yet?
```

```
0F39   9CEA            JNZ     Next
0F3B   90E3            JMP     Enter
                ;
       0000   TC1      =       0          ;Bit-time
       0004   TC2      =       4          ;Delay constants
                ;
       0F80   Scrat    =       0F80       ;Start of scratch area
       0000            .END
```

Digital Alarm Clock

```
              ;Outputs are held on when alarm
              ;time=Actual time, i.e. for one sec.
              ;
       010B   Crom     =       010B       ;Segment table
       0D00   Disp     =       0D00       ;Display address
       0F00   Ram      =       0F00
       0F10   Row      =       Ram+010
0000                   .=0F12
0F12                   .=.+1              ;Alarm time:hours
0F13                   .=.+1              ;Minutes
0F14                   .=.+1              ;Seconds
0F15                   .=.+1              ;Not used
0F16          Time:    .=.+4              ;Actual time
0F1A   76              .BYTE   076        ;Excess: Hours
0F1B   40              .BYTE   040        ;Minutes
0F1C   40              .BYTE   040        ;seconds
0F1D   20     Speed:   .BYTE   020        ;Speed
0F1E                   .=0F20
0F20   C401   Clock:   LDI     H(Crom)
0F22   37              XPAH    3
0F23   C40B            LDI     L(Crom)
0F25   33              XPAL    3
0F26   C40D   New:     LDI     H(Disp)
0F28   36              XPAH    2
0F29   C40D            LDI     L (Disp) +0D
0F2B   32              XPAL    2
0F2C   C40F            LDI     H(Time)
0F2E   35              XPAH    1
0F2F   C41A            LDI     L(Time)+4
0F31   31              XPAL    1
0F32   03              SCL
0F33   C405            LDI     5          ;Loop count
0F35   C8DA            ST      Row
0F37   C5FF   Again:   LD      @-1(1)
0F39   EC00            DAI     0
0F3B   C900            ST      (1)
0F3D   E904            DAD     +4(1)
0F3F   9804            JZ      Cs
0F41   9802            JZ      Cs         ;Equalize paths
0F43   9002            JMP     Cont
0F45   C900   Cs:      ST      (1)
```

```
0F47   C100         Cont:   LD     (1)
0F49   D40F                 ANI    0F
0F4B   01                   XAE
0F4C   C380                 LD     -128(3)  ;Get segments
0F4E   CE01                 ST     @+1(2)   ;Write to display
0F50   C440                 LDI    040
0F52   8F00                 DLY    00       ;Equalize display
0F54   C100                 LD     (1)
0F56   1C                   SR
0F57   1C                   SR
0F58   1C                   SR
0F59   1C                   SR
0F5A   01                   XAE
0F5B   C380                 LD     -128(3)
0F5D   CE02                 ST     @+2(2)   ;Leave a gap
0F5F   B8B0                 DLD    Row
0F61   9CD4                 JNZ    Again
0F63   C403                 LDI    3
0F65   C8AA                 ST     Row      ;Digit count
0F67   C400                 LDI    0
0F69   01                   XAE
0F6A   C5FF         Loop:   LD     @-1(1)
0F6C   E104                 XOR    +4(1)    ;Same time?
0F6E   58                   ORE
0F6F   01                   XAE
0F70   B89F                 DLD    Row
0F72   9CF6                 JNZ    Loop
0F74   01                   XAE
0F75   9803                 JZ     Alarm    ;Times tally
0F77   40                   LDE
0F78   9003                 JMP    Contin
0F7A   C407         Alarm:  LDI    07       ;All flags on
0F7C   08                   NOP             ;Pad out path
0F7D   07           Contin: CAS             ;Output to flags
0F7E   C4FD 16  C9          LDI    0FD      ;Pad out loop to
0F80   8F06 18  15          DLY    06       ;1/(100-speed) secs.
0F82   90A2                 JMP    New

       0000                 .END
```

Random Noise

```
                    ; Relocatable
                    ; Generates sequence 2↑15 bits long
                    ;
                            .=0F1E
0F1E                Line:   .=.+1                ;For random number
                    ;                            ;Must not be zero
0F20   C0FD         Noise:  LD     Line
0F22   1F                   RRL
0F23   C8FA                 ST     Line
0F25   C0F9                 LD     Line+1
```

```
0F27    1F          RRL
0F28    C8F6        ST      Line+1
0F2A    02          CCL             ;Ex-or of bits 1 and 2
0F2B    F402        ADI     02      ;In bit 3
0F2D    1E          RR              ;Rotate bit 3 to
0F2E    1E          RR              ;Bit 7
0F2F    1E          RR
0F30    D487        ANI     087     ;Put it in carry and
0F32    07          CAS             ;Update flags
0F33    90EB        JMP     Noise

        0000        .END
```

System

'Single Step', or SS, add the facility of being able to step through a program being debugged, executing it an instruction at a time, the next address and op-code being displayed after each step. SS is set up by storing the start address of the user program at OFF7 and OFF8. Then 'GO'ing to SS will cause the user program's start address and first instruction to be displayed.
Pressing 'MEM' then executes that instruction and displays the next one. Thus one can step through checking that jumps lead to the correct address and that the expected flow of control is achieved. If, in between steps, 'ABORT' is pressed, control is returned to the monitor and the contents of the registers from that point in the execution of the user program may be examined in memory where they are stored between steps:

OFF7	PCH	Program Counter
OFF8	PCL	
OFF9	P1H	Pointer 1
OFFA	P1L	
OFFB	P2H	Pointer 2
OFFC	P2L	
OFFD	A	Accumulator
OFFE	E	Extension Register
OFFF	S	Status Register

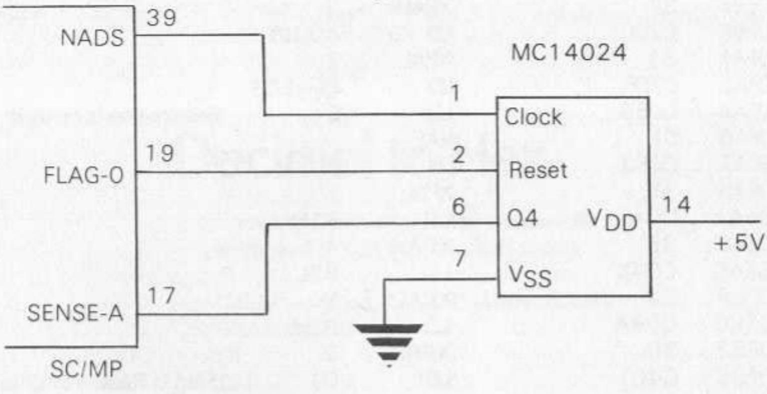
'GO'ing to the start of SS again will take up execution where it was left off. The values of the registers are taken from these locations so it is possible to alter them between steps.

The additional circuitry needed to implement the single step facility is shown in Fig. 1. A CMOS counter, clocked by the NADS signal from SC/MP, is reset from the SS program by a pulse at FLAG-0. After 8 NADS pulses it puts SENSE—A high; this will be the instruction fetch of the next instruction in the user's program, and an interrupt will be caused after that instruction has been executed. The interrupt returns control to SS ready for the next step. A TTL binary counter could be used in this circuit instead.
The 'Decimal to Hex' conversion program displays in hex the decimal number entered in at the keyboard as it is being entered. Negative numbers can be entered too, prefixed by 'MEM'.

e.g. 'MEM' '1' '5' '7' displays 'FF63'

'TERM' clears the display ready for a new number entry.
Any of the programs marked relocatable can be moved, without alteration, to a different start address and they will execute in exactly the same manner. The program 'Relocator' will move up to 256 bytes at a time from any start address to any destination address.
These two addresses and the number of bytes to be moved are specified in the 5 locations before the program. Since the source program and destination area may overlap, the order in which bytes are transferred is critical to avoid overwriting data not yet transferred, and so the program tests for this.

Fig. 1



Single Step

```
                ; Adds a facility for executing programs a
                ; Single instruction at a time, displaying
                ; The program counter and op-code
                ; After each step.
                ;
                ; To examine registers, abort and
                ; use the monitor in the usual way.
                ; To continue, go to 0F90.
                ;
0FF7    P3H     =       0FF7    ;For program to be
0FF8    P3L     =       0FF8    ;Single-stepped
0FF9    P1H     =       0FF9    ;Save user's registers:
0FFA    P1L     =       0FFA    ;(can be examined or
0FFB    P2H     =       0FFB    ;altered between
0FFC    P2L     =       0FFC    ;steps from monitor)
0FFD    A       =       0FFD
0FFE    E       =       0FFE
0FFF    S       =       0FFF
                ;
000C    ADL     =       12
000E    ADH     =       14
000D    Word    =       13
0F00    Ram     =       0F00
0140    Dispd   =       0140
                ;
                ;Program enter here
0000                    .=0F90
0F90    C86C    SS:     ST      A
0F92    C065            LD      P3L         ;Pick up user's program
0F94    33              XPAL    3           ;Address
0F95    C061            LD      P3H
0F97    37              XPAH    3
0F98    C7FF            LD      @-1(3)      ;Ready for jump
0F9A    9025            JMP     Ret
                        ;
```

```
0F9C    C20E    Step:   LD      ADH(2)
0F9E    37              XPAH    3
0F9F    C20C            LD      ADL(2)
0FA1    33              XPAL    3
0FA2    C7FF            LD      @-1(3)
0FA4    C059            LD      E           ;Restore user's context:
0FA6    01              XAE
0FA7    C052            LD      P1L
0FA9    31              XPAL    1
0FAA    C04E            LD      P1H
0FAC    35              XPAH    1
0FAD    C04E            LD      P2L
0FAF    32              XPAL    2
0FB0    C04A            LD      P2H
0FB2    36              XPAH    2
0FB3    C401            LDI     01          ;Flag 0 Resets counter
0FB5    07              CAS                 ;Put it high
0FB6    C048            LD      S
0FB8    D4FE            ANI     X'FE        ;Put flag 0 low
0FBA    07              CAS                 ;Start counting nads
0FBB    C041            LD      A
0FBD    05              IEN
0FBE    08              NOP                 ;Pad out to 8
0FBF    08              NOP
0FC0    3F              XPPC    3           ;Go to user's program
                ;Here on interrupt after one instruction
0FC1    C83B            ST      A           ;Save user's context
0FC3    40      Ret:    LDE
0FC4    C839            ST      E
0FC6    06              CSA
0FC7    C837            ST      S
0FC9    35              XPAH    1
0FCA    C82E            ST      P1H
0FCC    31              XPAL    1
0FCD    C82C            ST      P1L
0FCF    C40F            LDI     H(Ram)      ;Set P2-> Ram
0FD1    36              XPAH    2
0FD2    C828            ST      P2H
0FD4    C400            LDI     L(Ram)
0FD6    32              XPAL    2
0FD7    C824            ST      P2L
0FD9    C701            LD      @1(3)
0FDB    C300            LD      (3)         ;Get op-code
0FDD    CA0D            ST      Word(2)
0FDF    C401            LDI     H(Dispd)
0FE1    37              XPAH    3
0FE2    CA0E            ST      ADH(2)
0FE4    C812            ST      P3H         ;So can enter via 'SS'
0FE6    C43F            LDI     L(Dispd)-1
0FE8    33              XPAL    3
0FE9    CA0C            ST      ADL(2)
0FEB    C80C            ST      P3L
0FED    3F      No:     XPPC    3           ;Go to display routine
```

```
0FEE    90AC            JMP     Step        ;Command return so step
0FF0    90FB            JMP     No          ;Number return illegal

        0000            .END
```

Decimal to Hex

```
                ; Converts decimal number entered at
                ; keyboard to hex and displays result
                ;
                ; 'MEM' = minus, 'TERM' clears display
                ; (Relocatable)
        000C    ADL     =       0C
        000E    ADH     =       0E
        0F00    Ram     =       0F00
        015A    Dispa   =       015A
        0011    Count   =       011
        0012    Minus   =       012
        0013    Ltemp   =       013
                ;
0000                    .=0F50
0F50    C400    Dhex:   LDI     0
0F52    CA12            ST      Minus(2)
0F54    CA0E            ST      ADH(2)
0F56    CA0C            ST      ADL(2)
0F58    C401    Disp:   LDI     H(Dispa)
0F5A    37              XPAH    3
0F5B    C459            LDI     L(Dispa)-1
0F5D    33              XPAL    3
0F5E    3F              XPPC    3
0F5F    9028            JMP     Comd        ;Command key
0F61    C40A            LDI     10          ;Number in extension
0F63    CA11            ST      Count(2)    ;Multiply by 10
0F65    03              SCL
0F66    C212            LD      Minus(2)
0F68    01              XAE
0F69    60              XRE
0F6A    78              CAE
0F6B    01              XAE
0F6C    40              LDE                 ;Same as: LDI 0
0F6D    78              CAE                 ;         CAD 0
0F6E    01              XAE
0F6F    9002            JMP     Digit
0F71    C213    Addd:   LD      Ltemp(2)    ;Low byte of product
0F73    02      Digit:  CCL
0F74    F20C            ADD     ADL(2)
0F76    CA13            ST      Ltemp(2)
0F78    40              LDE                 ;High byte of product
0F79    F20E            ADD     ADH(2)
0F7B    01              XAE                 ;Put back
0F7C    BA11            DLD     Count(2)
0F7E    9CF1            JNZ     Addd
```

```
0F80    40              LDE
0F81    CA0E            ST      Adh(2)
0F83    C213            LD      Ltemp(2)
0F85    CA0C            ST      Adl(2)
0F87    90CF            JMP     Disp        ;Display result
0F89    E403    Comd:   XRI     3           ;'TERM'?
0F8B    98C3            JZ      Dhex        ;Restart if so
0F8D    C4FF            LDI     X'FF        ;Must be 'MEM'
0F8F    CA12            ST      Minus(2)
0F91    90C5            JMP     Disp
                ;
0F93                    .=0FFB
0FFB    0F00            .DBYTE  Ram         ;Set P2-> Ram
                ;
        0000            .END
```

Relocator

```
                ;Moves block of memory
                ;'From' = source start address
                ;'To' = destination start address
                ;'Length' = No of bytes
                ;(Relocatable)
                ;
        FF80    E       =       -128        ;Extension as offset
0000                    .=0F1B
                ;
0F1B            From:   .=.+2
0F1D            To:     .=.+2
0F1F            Length: .=.+1
                ;
0F20    C400    Entry:  LDI     0
0F22    01              XAE
0F23    03              SCL
0F24    C0F9            LD      To+1
0F26    F8F5            CAD     From+1
0F28    C0F4            LD      To
0F2A    F8F0            CAD     From
0F2C    1D              SRL
0F2D    9403            JP      Fgt         ;'From' greater than 'To'
0F2F    C0EF            LD      Length      ;Start from end
0F31    01              XAE
0F32    02      Fgt:    CCL
0F33    C0E8            LD      From+1
0F35    70              ADE
0F36    31              XPAL    1
0F27    C0E3            LD      From
0F39    F400            ADI     0
0F3B    35              XPAH    1
0F3C    02              CCL
0F3D    C0E0            LD      To+1
0F3F    70              ADE
```

```
0F40   32                XPAL   2
0F41   C0DB              LD     To
0F43   F400              ADI    0
0F45   36                XPAH   2
0F46   02                CCL
0F47   40                LDE
0F48   9C02              JNZ    Up
0F4A   C402              LDI    2
0F4C   78       Up:      CAE           ;i.e. subtract 1
0F4D   01                XAE           ;Put it in ext.
0F4E   C580     Move:    LD     E(1)
0F50   CE80              ST     @E(2)  ;Move byte
0F52   B8CC              DLD    Length
0F54   9CF8              JNZ    Move
0F56   3F                XPPC   3      ;Return

       0000              .END
```

Serial Data Transfers with SC/MP-ii

This application note describes a method of serial data input/output (I/O) data transfer using the SC/MP-II (ISP-8A/600) Extension Register. All data I/O is under direct software control with data transfer rates between 110 baud and 9600 baud selectable via software modification.

Data Output

Data to be output by SC/MP-II is placed in the Extension Register and shifted out through the SOUT Port using the Serial Input/Output Instruction (SIO). The Delay Instruction (DLY), in turn, creates the necessary delay to achieve the proper output baud rate. This produces a TTL-level data stream which can be used as is or can be level-shifted to an RS-232C level. Numerous circuits are available for level shifting. As an example, either a DS 1488 or an operational amplifier can be used. Inversion of the data stream, if needed, can be done either before the signal is converted or by the level shifter itself.

Data Input

Data input is received in much the same way as data is output. The Start Bit is sensed at the SIN Port and then received using the SIO Instruction and the DLY Instruction. After the Start Bit is received, a delay into the middle of the bit-time is executed. the data is then sensed at each full bit-time (the middle of the bit) until all data bits are received. If the data is at an RS-232C level, it must be shifted to a TTL level which SC/MP-II can utilize. This can be done with either a DS 1489 or an operational amplifier. If inversion if the data is necessary, it should be done before it is presented to the SIN Port.

Timing Considerations

Using the I/O routines presented in this application note, the user will be able to vary serial data transmission rates by simply changing the delay constants in each of the programs. Table 1 contains the delay constants needed for the various input baud rates. Table 2 contains the delay constants needed for the various output baud rates. Figure 1 is the outline used for Serial Data Input. Figure 2 is the routine used for Serial Data Output.

Baud Rate	Bit Time	HBTF	HBTC	BTF	BTC
110	9.09 ms	X'C3	X'8	X'92	X'11
300	3.33 ms	X'29	X'3	X'5E	X'6
600	1.67 ms	X'8A	X'1	X'20	X'3
1200	0.833ms	X'BB	X'0	X'81	X'1
2400	0.417ms	X'52	X'0	X'B2	X'0
4800	0.208ms	X'1F	X'0	X'4A	X'0
6400	0.156ms	X'12	X'0	X'30	X'0
9600	0.104ms	X'5	X'0	X'16	X'0

Table 1. Input Delay Constants (4 MHz SC/MP-II)

Baud Rate	Bit Time	BTF1	BTF2	BTC
110	9.09 ms	X'91	X'86	X'11
300	3.33 ms	X'5E	X'53	X'6
600	1.67 ms	X'1F	X'14	X'3
1200	0.833 ms	X'81	X'76	X'1
2400	0.417 ms	X'B2	X'A7	X'0
4800	0.208 ms	X'49	X'3E	X'0
6400	0.156 ms	X'2F	X'24	X'0
9600	0.104 ms	X'15	X'A	X'0

Table 2. Output Delay Constants (4 MHz SC/MP-II)

NOTES:

1. The Serial Data Output routine requires that the bit-count (BITCNT) in the program be set to the total number of data bits and stop bits to be used per character.

2. Two stop bits are needed for the 110 baud rate; all other baud rates need only one stop bit.

Serial Data Input

```
                       Title Recv, 'SERIAL DATA INPUT'

            0001  P1=1
            0002  P2=2
            0003  P3=3

                  ; Routine is called with a "XPPC P3" instruction

                  ; Data is received through the serial I/O Port.

                  ; Before executing routine, Pointer  2 should point
                  ; to one available location in R/W memory for a
                  ; counter.
                  ; On return from routine, data received will be in the
                  ; Accumulator and the Extension Register.

                  ; Delay Constants, user defined for desired Baud rate.
                  ; The following example is for 1200 Baud:

            00BB  HBTF    =       0BB      ; Half Bit time, Fine
            0000  HBTC    =       0        ; Half Bit time, Coarse
            0081  BTF     =       081      ; Full Bit Time, Fine
            0001  BTC     =       01       ; Full Bit time, Coarse

                  Search:
      0000  C408          LDI     08       ; Initialize Loop Counter
      0002  CA00          ST      (P2)     ; Save in memory
                  Again:
```

```
0004 C400          LDI    0        ;Clear Accumulator
0006 01            XAE             ; Clear E. Reg.
0007 19            S10             ;Look for Start Bit
0008 40            LDE             ; Bring into Acc.
0009 9CF9          JNZ    Again    ; If not zero, look again
000B C4BB          LDI    HBTF     ; Load Acc Half Bit time
000D 8F00          DLY    HBTC; Delay Half Bit time
000F 19            SIO             ; Check Input again to
0010 01            XAE             ; be sure of Start Bit
0011 9CF1          JNZ    Again    ; If not zero, was not
0013 C400          LDI    0        ; start B
0015 01            XAE
              Loop:
0016 C481          LDI    BTF      ; Load Bit time Fine
0018 8F01          DLY    BTC      ; Delay one Bit time
0001A19            SIO             ; Shift in Data Bit
001B BA00          DLD    (P2)     ; decrement loop counter
001D 9CF7          JNZ    Loop     ;Test for done
001F 40            LDE             ; Done, put data in acc.
0020 3F            XPPC   P3

     0000          END

AGAIN  0004   BTC     0001   BTF   0081   HBTC   0000
HBTF   00BB   LOOP    0016   P1    0001   P2     0002
P3     0003   SEARCH  0000*
```

Serial Data Output

```
                 TITLE XMIT, 'SERIAL DATA OUTPUT'

     0001 P1=1
     0002 P2=2
     0003 P3=3

          ; Routine is called with a ''XPPC P3'' instruction.

          ; Data is transmitted through Serial I/O Port.

          ; Before executing subroutine, pointer  2 should
          ; point to one available byte of R/W memory for a
          ; counter.
          ; Upon entry, character to be transmitted must be in
          ; the accumulator.

          ; Delay constants, user defined for desired baud rate.
          ; The following example is for 1200 baud:

     0081 BTF1    =    081    ; Bit time Fine, first loop
     0076 BTF2    =    076    ; Bit time Fine, second loop
     0001 BTC     =    01     ; Full Bit time, Coarse
```

```
                  ; Character Bit-count. This should be set for the
                  ; desired number of Data Bits and stop Bits.

           0009 BITCNT  =       9        ; 8 data and 1 Stop Bit

                Start:
      0000 01           XAE              ; Save data in E. Reg.
      0001 C400         LDI     0        ; Clear acc.
      0003 01           XAE              ; Put data in acc, clear E.
      0004 19           SIO              ; Send Start Bit
      0005 01           XAE              ; Put data in E. Reg.
      0006 C481         LDI     BTF1     ; Load Bit time Fine
      0008 8F01         DLY     BTC      ; Wait one Bit time
      000A C409         LDI     BITCNT   ; Set loop count for data
      000C CA00         ST      (P2)     ; and Stop Bit(s). Save
                Send:                    ; in count.
      000E 19           SIO              ; Send Bit
      000F 40           LDE
      0010 DC80         ORI     080      ; Set last Bit to 1
      0012 01           XAE              ; Put back in E. Reg.
      0013 C476         LDI     BTF2     ; Load Bit time Fine
      0015 8F01         DLY     BTC      ; Delay one Bit time
      0017 BA00         DLD     (P2)     ; decrement Bit counter
      0019 9CF3         JNZ     Send     ; If not done, loop back
      001B 3F           XPPC    P3       ; otherwise, return

           0000         END

BITCNT 0009   BTC    0001   BTF1   0081   BTF2   0076
P1     0001*  P2     0002   P3     0003   SEND   000E
START  000*
```

Games

The first two games are real-time simulations which provide a test of skill, and they can be adjusted in difficulty to suit the player's ability. The last two games are both tests of clear thinking and logical reasoning, and in the last one you are pitted against the microprocessor which tries to win.

'Moon Landing' simulates the landing of a spacecraft on the moon. The displays represent the control panel and give a continuously changing readout of altitude (3 digits), rate of descent (2 digits), and fuel remaining (1 digit). The object of the game is to touch down gently; i.e. to reach zero altitude with zero rate of descent. To achieve this you have control over the thrust of the rockets: the keys 1 to 7 set the thrust to the corresponding strength, but the greater the thrust the higher the rate of consumption of fuel. When the fuel runs out an 'F' is displayed in the fuel gauge, and the spacecraft will plummet to the ground under the force of gravity.

On reaching the moon's surface the display will freeze showing the velocity with which you hit the surface if you crashed, and the fuel remaining. Pressing 'TERM' will start a new landing.

The speed of the game is determined by the delay constants at OF38 and OF3A. The values given are suitable for a 1 MHz clock and they should be increased in proportion for higher clock rates. The initial values for the altitude, velocity, and fuel parameters are stored in memory at OF14 to OF1F and these can be altered to change the game.

'Duck Shoot' simulates ducks flying across the skyline. At first there is one duck, and it can be shot by hitting the key corresponding to its position: 7 = leftmost display, 0 = rightmost display. If you score a hit the duck will disappear; if you miss however, another duck will appear to add to you task.

The counter at OF1D varies the speed of flight and can be increased to make the game easier.

In 'Mastermind' the player tries to deduce a 'code' chosen by the machine. The code consists of four decimal digits, and pressing 'TERM' followed by 'MEM' causes the machine to choose a new code. The player makes guesses at the code which are entered at the keyboard. Pressing 'GO' then causes the machine to reveal two pieces of information, which are displayed as two digits:

(1) The number of digits in the guess which are correct and in the right position, (known as 'Bulls') and
(2) the number of digits correct but in the wrong position, (known as 'Cows').

For example, suppose that the machine's code was '6678'. The following guesses would then score as shown:

1234	0–0	1278	2–0
7812	0–2	7687	1–2

Subsequent guesses are entered in a similar way, and the player tries to deduce the code in as few attempts as possible.

'Silver Dollar Game' is traditionally played with a number of coins which are moved by the players in one direction along a line of squares. In his turn a player must move a coin to the right across as many unoccupied

squares as he wishes. The player first unable to move—when all the coins have reached the right-hand end of the line—loses, and the other player takes the coins!

In this version of the game the coins are represented by vertical bars moving along a dashed line. There are five coins numbered, from right to left, 1 to 5. The player makes his move by pressing the key corresponding to the number of the coin he wishes to move, and each press moves the coin one square along to the right. The machine plays against you, and pressing 'MEM' causes it to make its move. Note that the machine will refuse to move in its turn unless you have made a legal move in your turn. 'TERM' starts a new game.

The machine allows you to take first move and it is possible to win from the starting position given, though this is quite difficult. The five numbers in locations 0F13 to 0F17 determine the starting positions of each coin and these can be altered to any other values in the range 00 to 0F provided they are in ascending order.

Moon Landing

```
                ; Land a rocket on the moon
                ; Display shows altitude-velocity-fuel
                ; Keys 1-7 control the thrust
                ;
        0005    Grav    =       5         ;Force of gravity
        0D00    Disp    =       0D00      ;Display address
        010B    Crom    =       010B      ;Segment table
        FF80    E       =       -128      ;Extension as offset
        FFE3    Row     =       Ret-0F03  ;Ram offsets
        FFE4    Count   =       Ret-0F04
                ;Variables
0000                    .=0F05
0F05            Save:   .=.+1
0F06            H1:     .=.+1
0F07            L1:     .=.+1
0F08            Alt:    .=.+3             ;Altitude
0F0B            Vel:    .=.+3             ;Velocity
0F0E            Accn:   .=.+2             ;Acceleration
0F10            Thr:    .=.+2             ;Thrust
0F12            Fuel:   .=.+2             ;Fuel left
                ;Original values
0F14    08      Init:   BYTE    08,050,0;Altitude=850
        50
        00
0F17    99              .BYTE   099,080,0;Velocity=-20
        80
        00
0F1A    99              .BYTE   099,098 ;Acceleration=-2
        98
0F1C    00              .BYTE   0,02    ;Thrust=2
        02
0F1E    68              .BYTE   058,0   ;Fuel=5
        00
```

```
                ;Subroutine to display AC as two digits
0F20   3E       Ret:    XPPC    2          ;P2 contains 0F20
0F21   C8E3     Disp:   ST      Save
0F23   C401             LDI     H(Crom)
0F25   35               XPAH    1
0F26   C8DF             ST      H1         ;Run out of pointers
0F28   C40B             LDI     L(Crom)
0F2A   31               XPAL    1
0F2B   C8DB             ST      L1
0F2D   C0D7             LD      Save
0F2F   02               CCL
0F30   D40F             ANI     0F
0F32   01       Loop:   XAE
0F33   C180             LD      E(1)
0F35   CF01             ST      @+1(3)
0F37   C400             LDI     0          ;Delay point
0F39   8F02             DLY     2          ;Determines speed
0F3B   C0C9             LD      Save
0F3D   1C               SR
0F3E   1C               SR
0F3F   1C               SR
0F40   1C               SR
0F41   01               XAE
0F42   06               CSA
0F43   03               SCL
0F44   94ED             JP      Loop       ;Do it twice
0F46   C400             LDI     0
0F48   CF01             ST      @+1(3)     ;Blank between
0F4A   C0BB             LD      H1         ;Restores P1:
0F4C   35               XPAH    1
0F4D   C0B9             LD      L1
0F4F   31               XPAL    1
0F50   90CE             JMP     Ret        ;Return
                ;Main moon-landing program
0F52   C40F     Start:  LDI     H(Init)
0F54   35               XPAH    1
0F55   C414             LDI     L(Init)
0F57   31               XPAL    1
0F58   C40F             LDI     H(Ret)
0F5A   36               XPAH    2
0F5B   C420             LDI     L(Ret)
0F5D   32               XPAL    2
0F5E   C40C             LDI     12
0F60   CAE4             ST      Count(2)
0F62   C10B     Set:    LD      +11(1)
0F64   CDFF             ST      @-1(1)
0F66   BAE4             DLD     Count(2)
0F68   9CF8             JNZ     Set
                ;Main loop
0F6A   C40C     Again:  LDI     H(Disp)-1
0F6C   37               XPAH    3
0F6D   C4FF             LDI     L(Disp)-1
0F6F   33               XPAL    3
0F70   C401             LDI     1
0F72   CAE4             ST      Count(2)
```

```
0F74   C506             LD     @+6(1)   ;P1-> Vel+2
0F76   9404             JP     Twice    ;Altitude positive?
0F78   C504             LD     @+4(1)   ,P1-> Thr+1
0F7A   9032             JMP    Off      ;Don't update
0F7C   C402   Twice:    LDI    2        ;Update velocity anc
0F7E   CAE3             ST     Row(2)   ;Then altitude....
0F80   02               CCL
0F81   C5FF   Dadd:     LD     @-1(1)
0F83   E902             DAD    +2(1)
0F85   C900             ST     (1)
0F87   BAE3             DLD    Row(2)
0F89   9CF6             JNZ    Dadd
0F8B   C102             LD     +2(1)
0F8D   9402             JP     Pos      ;Gone negative?
0F8F   C499             LDI    X'99
0F91   EDFF   Pos:      DAD    @-1(1)
0F93   C900             ST     (1)
0F95   BAE4             DLD    Count(2)
0F97   94E3             JP     Twice
0F99   C50C             LD     @12(1)   ;P1-> Alt
0F9B   AAE3             ILD    Row(2)   ;Row:=1
0F9D   03               SCL
0F9E   C5FF   D sub:    LD     @-1(1)   ;Fuel
0FA0   F9FE             CAD    -2(1)    ;Subtract thrust
0FA2   C900             ST     (1)
0FA4   08               NOP
0FA5   BAE3             DLD    Row(2)
0FA7   94F3             JP     Dsub
0FA9   06               CSA             ;P1-> Fuel now
0FAA   9402             JP     Off      ;Fuel run out?
0FAC   9004             JMP    Accns
0FAE   C400   Off:      LDI    0
0FB0   C9FF             ST     -1(1)    ;Zero thrust
0FB2   C1FF   Accns:    LD     -1(1)
0FB4   03               SCL
0FB5   EC94             DAI    099-Grav
0FB7   C9FD             ST     -3(1)    ;Accn+1
0FB9   C499             LDI    X'99
0FBB   EC00             DAI    0
? 0FBC C9FC             ST     -4(1)    ;Accn
0FBF   C100   Dispy:    LD     (1)      ;Fuel
0FC1   3E               XPPC   2        ;Display it OK
0FC2   C1F9             LD     -7(1)    ;Vel
0FC4   940A             JP     Posv
0FC6   C499             LDI    X'99
0FC8   03               SCL
0FC9   F9FA             CAD    -6(1)    ;Vel+1
0FCB   03               SCL
0FCC   EC00             DAI    0
0FCE   9002             JMP    STO
0FD0   C1FA   Posv:     LD     -6(1)    ;Vel+1
0FD2   3E     Sto:      XPPC   2        ;Display velocity
0FD3   C1F7             LD     -9(1)    ;Alt+1
```

```
0FD5   3E              XPPC   2          ;Display it
0FD6   C7FF            LD     @-1(3)     ;Get rid of lank
0FD8   C5F6            LD     @-10(1);P1-> Alt now
0FDA   3E              XPPC   2
0FDB   C40A            LDI    10
0FDD   CAE4            ST     Count(2)
0FDF   C7FF    Toil:   LD     @-1(3)     ;Key pressed?
0FE1   940A            JP     Press      ;Key 0-7?
0FE3   E4DF            XRI    X'DF       ;Command Key?
0FE5   9A31            JZ     Start(2)   ;Begin again if so
0FE7   BAE4            DLD    Count(2)
0FE9   9CF4            JNZ    Toil
0FEB   9249            JMP    Again(2)   ;Another circuit
0FED   C109            LD     +9(1)      ;Thr + 1
0FEF   9803            JZ     Back       ;Engines stopped?
0FF1   33              XPAL   3          ;Which row?
0FF2   C909            St     +9(1)      ;Set thrust
0FF4   9249    Back:   JMP    Again(2)   ;Carry on counting
       0000            END
```

Duck Shoot

```
                ; Shoot Ducks flying display
                ; By hitting key with number corresponding
                ; To their position: 7 = Leftmost,
                ; 0 = Rightmost.
                ; If you miss, another duck appears
                ; (Relocatable)
                Duck    =      061       ;Segment pattern
                Disp    =      0D00      ;Display address
0000                    . = 0F0F
0F0F            Row:    . = . + 1        ;Bits set = ducks
0F10            Count:  . = . + 1
0F11            Sum:    . = . + 1        ;Key pressed
                ;
0F12   C40D     Shoot:  LDI    H(Disp)
0F14   35               XPAH   1
0F15   C400             LDI    L(Disp)
0F17   31               XPAL   1
0F18   C401             LDI    1         ;Start with 1 duck
0F1A   C8F4             ST     Row
0F1C   C410     React:  LDI    16        ;Speed of flight,
0F1E   C8F1             ST     Count     ;Smaller = harder
0F20   C400             LDI    0
0F22   C8EE             ST     Sum
0F24   C408     Shift:  LDI    8         ;Move ducks this time
0F26   01       Ndig:   XAE
0F27   C0E7             LD     Row
0F29   1E               RR
0F2A   C8E4             ST     Row
0F2C   9404             JP     No
```

```
0F2E   C461             LDI    Duck
0F30   9002             JMP    Go
0F32   C400     No:     LDI    0            ;No duck
0F34   C980     Go:     ST     -128(1)      ;E as offset
0F36   8F01             DLY    01           ;Shine digit
0F38   C0D8             LD     Sum
0F3A   9C0E             JNZ    Nok          ;Key already pressed
0F3C   C180             LD     -128(1)      ;Test for key
0F3E   E4FF             XRI    0FF
0F40   9808             JZ     Nok          ;No key
0F42   C8CE             ST     Sum
0F44   C0CA             LD     Row
0F46   E480             XRI    080
0F48   C8C6             ST     Row          ;Change top bit
0F4A   40       Nok:    LDE
0F4B   03               SCL
0F4C   FC01             CAI    1            ;Subtract 1
0F4E   94D6             JP     Ndig         ;Do next digit
0F50   B8BF             DLD    Count
0F52   98C8             JZ     React        ;Start new position
0F54   C407             LDI    7
0F56   90CE             JMP    Ndig         ;Another sweep
       0000             .END
```

Mastermind

```
       0F00     Ram     =      0F00
       0D00     Disp    =      0D00         ;Display address
       010B     Crom    =      010B         ;Hex to segment table
       011B     Adr     =      011B         ;'Make 4 digit address'
       015A     Dispa   =      015A         ;'Address to segments'
                ;       Variables in RAM
       0000     Dl      =      0
       0002     D3      =      2
       0004     Adll    =      4
       000C     Adl     =      12
       000E     Adh     =      14
       000F     Ddta    =      15
       0010     Row     =      16
       0011     Next    =      17
       0014     Key     =      20
                ;       Begin at OFIC
0000                    .=OFIC
0F1C   C400     Start:  LDI    0
0F1E   C8ED             ST     ADL
0F20   C8ED             ST     ADH
0F22   32               XPAL   2
0F23   C40F             LDI    OF
0F25   36               XPAH   2
                ;       Choose  random   number
0F26   C401             LDI    H(Crom)
0F28   37               XPAH   3
```

```
0F29  C40B            LDI    L(Crom)
0F2B  33              XPAL   3
0F2C  C404  No Key:   LDI    04
0F2E  CA10            ST     Row(1)
0F30  C40F            LDI    H(digits)
0F32  35              XPAH   1
0F33  C414            LDI    L(Digits)
0F35  31              XPAL   1
0F36  03              SCL
0F37  C104  Incr:     LD     +4(1)
0F39  EC90            DAI    090
0F3B  C904            ST     +4(1)
0F3D  D40F            ANI    0F
0F3F  01              XAE
0F40  C380            LD     -128(3)
0F42  CD01            ST     @+1(1)
0F44  BA10            DLD    Row(2)
0F46  9CEF            JNZ    Incr
0F48  C40D            LDI    H(Disp)
0F4A  35              XPAH   1
0F4B  C400            LDI    L(Disp)
0F4D  31              XPAL   1
0F4E  C103            LD     3(1)       ;Key pressed?
0F50  E4FF            XRI    0FF
0F52  98D8            JZ     No key
                  ,   Enter your guess
0F54  C4FF  Clear:    LDI    0FF
0F56  CA0F            ST     Ddta(2)
0F58  C400            LDI    0
0F5A  CA00            ST     DL(2)
0F5C  CA02            ST     D3(2)
0F5E  02    Nchar:    CCL
0F5F  C401            LDI    H(Dispa)
0F61  37              XPAH   3
0F62  C459            LDI    L(Dispa)-1
0F64  33              XPAL   3
0F65  3F              XPPC   3          ;Jump to subroutine
0F66  900B            JMP    COMD       ;Command key return
0F68  40              LDE               ;Number key return
0F69  F4F6            ADI    0F6
0F6B  94F1            JP     Nchar      ;Ignore digits > 9
0F6D  C41A            LDI    L(Adr)-1
0F6F  33              XPAL   3
0F70  3F              XPPC   3
0F71  90E5            JMP    Blank      ;Get next digit
0F73  E403  Comd:     XRI    03         ;term?
0F75  9A1B            JZ     Start(2)   ;If so-new game
0F77  E405            XRI    05         ;Go?
0F79  9CD9            JNZ    Clear      ;Ignore if not
                  ,   Work out answer to guess
0F7B  C40B  Go:       LDI    L(Crom)
0F7D  CA00            ST     DL(2)
0F7F  CA02            ST     D3(2)
0F81  C40F  Bulls:    LDI    H(Key)
```

```
0F83   35             XPAH   1
0F84   C414           LDI    L(Key)
0F86   31             XPAL   1
0F87   C480           LDI    080
0F89   01             XAE
0F8A   C404           LDI    04           ;No. of digits
0F8C   CA11           ST     Next(2)
0F8E   C1F0   Bull 2: LD     Adll-Key(1)
0F90   E501           XOR    @+1(1)
0F92   9C0C           JNZ    Nobul
0F94   AA02           ILD    DH(2)
0F96   C1FF           LD     -1(1)
0F98   58             ORE                 ;Set negative
0F99   C9FF           ST     -1(1)
0F9B   C1EF           LD     Adll-Key-1(1)
0F9D   58             ORE
0F9E   C9EF           ST     Adll-Key-1(1)
0FA0   BA11   fBobul: DLD    Next(2)
0FA2   9CEA           JNZ    Bull 2
0FA4   C404   Cows:   LDI    04
0FA6   CA11           St     Next(2)      ;P1 points to Key + 4
0FA8   C404   Nerow:  LDI    04
0FAA   CA10           ST     Row(2)
0FAC   C40F           LDI    04
0FAA   CA10           ST     Row(2)
0FAC   C40F           LDI    H(Adll)
0FAE   37             XPAH   3
0FAF   C408           LDI    L(Adll) + 4
0FB1   33             XPAL   3
0FB2   C5FF           LD     @-1(1)
0FB4   940A           JP     Try          ;Already counted as bull?
0FB6   BA11   Nocow:  DLD    Next(2)      ;Yes
0FB8   9CEE           JNZ    Nerow
0FBA   9013           JMP    Finito
0FBC   BA10   Notry:  DLD    Row(2)
0FBE   98F6           JZ     Nocow
0FC0   C100   Try:    LD     (1)
0FC2   E7FF           XOR    @-1(3)       :Same?
0FC4   9CF6           JNZ    Notry
0FC6   AA00           ILD    DL(2)
0FC8   C300           LD     (3)
0FCA   58             ORE
0FCB   CB00           ST     (3)
0FCD   90E7           JMP    Nocow
              ; Now unset top bits of Key
0FCF   C404   Finito: LDI    04
0FD1   CA11           ST     Next(2)
0FD3   C100   Unset:  LD     (1)
0FD5   D47F           ANI    07F
0FD7   CD01           ST     @+1(1)
0FD9   BA11           DLD    Next(2)
0FDB   9CF6           JNZ    Unset        ;All done?
```

```
                ;Set up segments of result
0FDD   C401               LDI     H(Crom)
0FDF   35                 XPAH    1
0FE0   C200               LD      DL(2)     ;L(Crom) + Cows
0FE2   31                 XPAL    1
0FE3   C100               LD      (1)       ;Segments
0FE5   CA00               ST      DL(2)
0FE7   C202               LD      D3(2)     ;L(Crom) + Bulls
0FE9   31                 XPAL    1
0FEA   C100               LD      (1)       ;Segments
0FEC   CA02               ST      D3(2)
0FEE   C4FF               LDI     OFF
0FF0   CA0F               ST      Ddta(2)
0FF2   925D               JMP     Nchar(2)  ;Display result
                ;
       0000               .END
```

Silver Dollar Game

```
                ; Machine plays against you in moving five
                ; 'Silver Dollars' along a track
                ; Player unable to move loses
0000                      =0F12
                ; Starting position: Must be ascending order
0F12   FF       Start:    .BYTE   OFF
0F13   03                 .BYTE   03
0F14   05                 .BYTE   05
0F15   08                 .BYTE   08
0F16   09                 .BYTE   09
0F17   0F                 .BYTE   0
       0F00     Ram       =       0F00
0F18            Pos:      .=.+6             ;Current position
       0024     Count     =       024       ;Ram offsets:
       0025     Key       =       025       ;For key last pressed
       0026     Init      =       026       ;Zero
       0185     Kybd      =       0185      ;In monitor
       0080     E         =       -128      ;Extension reg.
                ;
0F1E                      .=0F28
0F28   C40F     Begin:    LDI     H(Ram)
0F2A   36                 XPAH    2
0F2B   C400               LDI     L(Ram)
0F2D   32                 XPAL    2
0F2E   C40F               LDI     H(Pos)
0F30   35                 XPAH    1
0F31   C418               LDI     L(Pos)
0F33   31                 XPAL    1
0F34   C406               LDI     6
0F36   CA24               ST      Count (2)
0F38   C1FA     Setup:    LD      -6(1)     ;Transfer start to pos
0F3A   CD01               ST      @+1(1)
0F3C   BA24               DLD     Count(2)
```

```
0F3E  9CF8            JNZ    Count(2)
0F40  C400  Ymove:    LDI    0            ;You go first!
0F42  CA25            ST     Key(2)       ;Clear key store
            ;Generate display from Pos
0F44  C40F  Disp:     LDI    H(Pos)
0F46  35              XPAH   1
0F47  C419            LDI    L(Pos) + 1
0F49  31              XPAL   1
0F4A  C409            LDI    9
0F4C  01    Clear:    XAE                 ;Clear Display buffer
0F4D  C408            LDI    08           ;Underline
0F4F  CA80            ST     E(2)
0F51  40              LDE
0F52  FC01            CAI    1
0F54  94F6            JP     Clear
0F56  C405            LDI    5
0F58  CA24            ST     Count(2)
0F5A  C501  Npos:     LD     @ + 1(1)
0F5C  1E              RR
0F5D  940B            JP     Even
0F5F  D47F  Odd:      ANI    07F
0F61  01              XAE
0F62  C280            LD     E(2)
0F64  DC30            ORI    030          ;Segments E & F
0F66  CA80            ST     E(2)
0F68  9007            JMP    Cont
0F6A  01    Even:     XAE
0F6B  C280            LD     E(2)
0F6D  DC06            ORI    06           ;Segments B & C
0F6F  CA80            ST     E(2)
0F71  BA24  Cont:     DLD    Count (2)
0F73  9CE5            JNZ    Npos
            ;Display current position
0F75  C401  Show:     LDI    H(Kybd)
0F77  37              XPAH   3
0F78  C484            LDI    L(Kybd)-1
0F7A  33              XPAL   3
0F7B  3F              XPPC   3
0F7C  902A            JMP    Coma         ;Command key
0F7E  40              LDE
0F7F  98F4            JZ     Show
0F81  03              SCL
0F82  FC06            CAI    6            ;1-5 allowed
0F84  94EF            JP     Show
0F86  C40F            LDI    H(Pos)
0F88  35              XPAH   1
0F89  C418            LDI    L(Pos)
0F8B  02              CCL
0F8C  70              ADE
0F8D  31              XPAL   1
0F8E  C100            LD     (1)
0F90  02              CCL
0F91  F4FF            ADI    —1
```

```
0F93   02             CCL
0F94   F9FF           CAD    -(1)
0F96   9402           JP     Fine 2     ;Valid move
0F98   90DB           JMP    Show
0F9A   C225  Fine 2:  LD     Key(2)
0F9C   9C03           JNZ    Firstn
0F9E   40             LDE
0F9F   CA25           ST     Key(2)     ;First key press
0FA1   60    Firstn:  XRE               ;Not first press
0FA2   9E43           JNZ    Disp(2)    ;not allowed
0FA4   B900           DLD    (1)        ;Make move
0FA6   9243           JMP    Disp(2)    ;Display result
0FA8   C225  Coma:    LD     Key(2)     ;Mem pressed
0FAA   9A43           JZ     Disp(2)    ;You haven't moved!
0FAC   C403  Go:      LDI    3
0FAE   CA24           ST     Count(2)
0FB0   C40F           LDI    H(Pos)
0FB2   35             XPAH   1
0FB3   C418           LDI    L(Pos)
0FB5   31             XPAL   1
0FB6   C400           LDI    0
0FB8   01             XAE
0FB9   C101  Try:     LD     +1(1)
0FBB   02             CCL
0FBc   FD02           CAD    @+2(1)
0FBE   C904           ST     4(1)
0FC0   60             XRE               ;Keep nim sum
0FC1   01             XAE
0FC2   BA24           DLD    Count(2)
0FC4
0FC4   9CF3           JNZ    Try
0FC6   40    Solve:   LDE
0FC7   980E           JZ     Nogo       ;Safe position
0FC9   E100           XOR    (1)
0FCB   03             SCL
0FCC   FD02           CAD    @+2(1)
0FCE   94F6           JP     Solve
0FD0   02             CCL
0FD1   F1F9           ADD    -7(1)      ;Make my move
0FD3   C9F9           ST     -7(1)
0FD5   923F           JMP    Ymove(2)   ;Now you, good luck!
0FD7   C405  Nogo:    LDI    05
0FD9   CA24           ST     Count(2)   ;Make first move
0FDB   C5FF  No:      LD     @-1(1)
0FDD   02             CCL
0FDE   F4FF           ADI    -1
0FE0   02             CCL
0FE1   F9FF           CAD    -1(1)
0FE3   9406           JP     Fine
0FE5   BA24           DLD    Count(2)
0FE7   9CF2           JNZ    No
0FE9   9307           JMP    +7(3)      ;i.e. Abort—I lose
0FEB   B900  Fine:    DLD    (1)        ;Make my move
0FED   923F           JMP    Ymove(2)   ;now you chum.
       0000           .END
```

Music

The 'Function Generator' produces a periodic waveform by outputting values from memory cyclically to a D/A converter. It uses the 8-bit port B of the RAM I/O chip to interface with the D/A, and Fig. 1 shows the wiring connections. The D/A chosen is the Ferranti ZN425E, a low-cost device with a direct voltage output.

Any waveform can be generated by storing the appropriate values in memory. The example given was calculated as an approximation to a typical musical waveform.

'Music Box' plays tunes stored in memory in coded form. The output can be taken from one of the flag outputs. Each note to be played is encoded as one byte. The lower 5 bits determine the frequency of the note, as follows:

Rest	A	A#	B	C	C#	D	D#	E	F	F#	G	G#
00	01	02	03	04	05	06	07	08	09	0A	0B	0C
	0D	0E	0F	10	11	12	13	14	15	16	17	18

There are two octaves altogether.

The top three bits of the byte give the duration of the note, as follows:

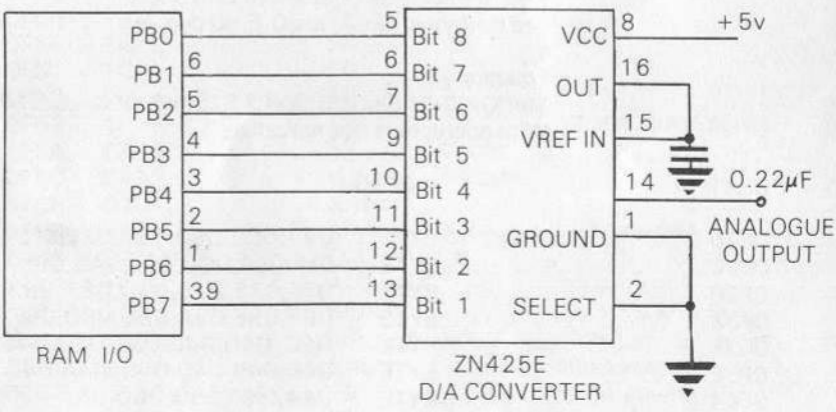
Relative Duration:	1	2	3	4	5	6	7	8
	00	20	40	60	80	A0	C0	E0

Thus for any specific note required the duration parameter and frequency parameter should be added together. A zero byte is reserved to specify the end of the tune.

To slow down the tempo locations 0F58 and 0F59 should be altered to D4FC (ANI X'FC).

The program uses two look-up tables, one giving the time-constant for a delay instruction determining the period of each note and the other giving the number of cycles required for the basic note duration.

'Organ' generates a different note for each key of the keyboard by using the key value as the delay parameter in a timing loop. Great skill is needed to produce tunes on this organ.



Function Generator

```
                    ; Generates arbitrary waveform by outputting
                    ; values to D/A Converter.
                    ; uses Ram I/O chip. (Relocatable).
                    ;
                    Portb       =       0E21
                    Ext         =       -128    ;Extension as offset
                    ;
0000                            .=0E80          ;Start of Ram in Ram/IO
0E80   C40F         Start:      LDI     H(Endw)
0E82   36                       XPAH    2
0E83   C448                     LDI     L(Endw)
0E85   32                       XPAL    2       ;P2-> End of waveform
0E86   C40E                     LDI     H(Portb)
0E88   35                       XPAH    1
0E89   C421                     LDI     L(Portb)
0E8B   31                       XPAL    1
0E8C   C4FF                     LDI     X'FF    ;All bits as outputs
0E8E   C902                     ST      +2(1)   ;Output definition B
0E90   C4D8         Reset:      LDI     -Npts
0E92   02                       CCL
0E93   01           Next:       XAE
0E94   C280                     LD      E(2)    ;Get next value
0E96   C900                     ST      (1)     ;Send to D/A
0E98   40                       LDE
0E9A   F401                     ADI     1       ;Point to next value
0E9C   98F3                     JZ      Reset   ;New sweep
0E9E   04                       DINT            ;Equalize paths
0E9F   90F3                     JMP     Next    ;Next point
                    ;
                    ;
                    ; Sample waveform of 40 points
                    ; Fundamental amplitude 1
                    ; 2nd Harmonic amplitude 0.5 zero phase
                    ; 3rd Harmonic amplitude 0.5 90 deg. lag.
                    ;
                    ; Equation is:
                    ; Sin(X)+0.5*Sin(2.0*X)40.5*Sin(3.0*X-0.5*PI)
                    ; With appropriate normalization
                    ;
0EA1                            .=0F20
                    ;
0F20                Wave:       .BYTE   077,092,0B0,0CB,0E1,0ED
0F26                            .BYTE   0EF,0E6,0D5,0BE,0A5,08E
0F2C                            .BYTE   07F,077,076,07D,087,092
0F32                            .BYTE   09B,09E,09A,090,080,06F
0F38                            .BYTE   05C, 04D,042,03D,03D,040
0F3E                            .BYTE   046,04B,04D,04D, 04A,046
0F44                            .BYTE   044,047,050,060
         0F48       Endw        =
         0028       NPTS        =       Endw-wave ;No. of points
         0000                   END
```

Music Box

```
                    ; Plays a tune stored in memory
                    ; 1 Byte per note
                    ; top 3 bits = duration (00-E0) = 1 to 8 units
                    ; bottom 5 bits = note (01-18) = 2 octaves
                    ;
0000                         . = 0F12
                    ;Table of notes
0F12                Scale:   .BYTE   0                 ;Silence
0F13                         .BYTE   0FF,0EC,0DB,0CA,0BB,0AC
0F19                         .BYTE   09E,091,085,079,06E,063
0F1F                         .BYTE   059,050,047,03F,037,030
0F25                         .BYTE   029,022,01C,016,011,00C
                    ;Table of cycles per unit time
0F2B                         .BYTE   044,048,04C,051,055,05B
0F31                         .BYTE   060,066,06C,072,079,080
0F37                         .BYTE   088,090,098,0A1,0AB,0B5
0F3D                         .BYTE   0C0,0CB,0D7,0E4,0F2,0FF
                    ;Program now:
0F43                Cycles:  . = . + 1
0F44                Count:   . = . + 1
                    ;
0F45     3F         Stop:    XPPC    3                 ;'Go, 'term', to play again
                    ;
0F46     C40F       Begin:   LDI     H(Scale)
0F48     35                  XPAH    1
0F49     C40F                LDI     H(Tune)
0F4B     36                  XPAH    2
0F4C     C490                LDI     L(Tune)
0F4E     32                  XPAL    2                 ;P2 points to tune
0F4F     C601       Play:    LD      @+1(2)            ;Get next note code
0F51     01                  XAE                       ;Save in ext.
0F52     40                  LDE
0F53     98F0                JZ      Stop              ;Zero = terminator
0F55     1C                  SR
0F56     1C                  SR
0F57     1C                  SR
0F58     1C                  SR
0F59     1C                  SR                        ;Shift duration down
0F5A     C8E9                ST      Count
0F5C     C412                LDI     L(Scale)
0F5E     01                  XAE
0F5F     D41F                ANI     X'1F              ;Get note part
0F61     02                  CCL
0F62     70                  ADE                       ;no carry out
0F63     31                  XPAL    1                 ;Point P1 to note
0F64     C100                LD      (1)               ;Note
0F66     01                  XAE                       ;Put it in ext.
0F67     C118       Hold:    LD      +24(1)            ;Cycle count
0F69     C8D9                ST      Cycles
0F6B     40         Peal:    LDE
```

```
0F6C   9C04             JNZ     Sound      ;Zero = silence
0F6E   8F80             DLY     X'80       ;Unit gap
0F70   9011             JMP     More
0F72   8F00   Sound:    DLY     X'00
0F74   06               CSA
0F75   E407             XRI     X'07       ;Change flags
0F77   07               CAS
0F7B   B8CA             DLD     Cycles
0F7A   9807             JZ      More
0F7C   08               NOP                ;Equalize paths to
0F7D   C410             LDI     X'10       ;Prevent clicks in
0F7F   8F00             DLY     X'00       ;Sustained notes
0F81   90E8             JMP     Peal
0F83   B8C0   More:     DLD     Count
0F85   94E0             JP      Hold
0F87   8F20             DLY     X'20       ;Gap between notes
0F89   90C4             JMP     Play       ;Get next note
              ;
0F8B                    . = 0F90
0F90          Tune:     .BYTE   02D,02D,02F,04C,00D,02F
0F96                    .BYTE   031,031,032,051,00F,02D,
0F9C                    .BYTE   02F,02D,02C,02D,00D,00F
0FA2                    .BYTE   011,012,034,034,034,054,
0FA8                    .BYTE   012,031,032,032,032,052,
0FAE                    .BYTE   011,02F,031,012,011,00F
0FB4                    .BYTE   00D,051,012,034,016,032
0FBA                    .BYTE   071,06F,08D,0

       0000             .END
```

Organ

```
              ; Each key on the keyboard generates a
              ; Different note (though the scale is
              ; Somewhat unconventional!)Relocatable.
              ;
                        . = 0F1F
0F1F          Count:    . = . + 1
       0D00   Disp:     =       0D00       ;Display & keyboard
              ;
0F20   C40D   Enter:    LDI     H(Disp)
0F22   35               XPAH    1
0F23   C400   New:      LDI     L(Disp)
0F25   31               XPAL    1
0F26   C408             LDI     08
0F28   C8F6             ST      Count      ;Key row
0F2A   C501   Again:    LD      @+1(1)
0F2C   E4FF             XRI     0FF        ;Key pressed?
0F2E   9808             JZ      No
0F30   8F00             DLY     00         ;Delay with AC = key
0F32   06               CSA
0F33   E407             XRI     07         ;Change flags
```

```
0F35   07             CAS
0F36   90EB           JMP    New
0F38   B8E6    No:    DLD    Count
0F3A   9CEE           JNZ    Again
0F3C   90E5           JMP    New

       0000           .END
```

Miscellaneous

'Message' gives a moving display of segment arrangements according to the contents of memory locations from 'Text' downwards until an 'end-of-text' character with the top bit set (e.g. 080). Each of the bits 0-6 of the word in memory corresponds, respectively, to the seven display segments a-g; if the bit is set, the display segment will be lit. Most of the letters of the alphabet can be formed from combinations of the seven segments: e.g. 076 corresponds to 'H', 038 to 'L', etc. The speed with which the message moves along the display depends on the counter at OF2D. If the first and last 7 characters are the same, as in the sample message given, the text will appear continuous rather than jumping from the end back to the start.

The 'Reaction Timer' gives a readout, in milliseconds, of the time taken to respond to an unpredictable event. To reset the timer the 'O' key should be pressed. After a random time a display will flash on. The program then counts in milliseconds until the 'MEM' key is pressed, when the time will be shown on the display.

The execution time of the main loop of the program should be exactly one millisecond, and for different clock rates the delay constants will have to be altered:

Rate	Location:	OF2A	OF37	OF39
1 MHz		07D	0A8	00
2 MHz		0FA	0A1	01
4 MHz		0FF	093	03

The 'Self-Replicating Program' makes a copy of itself at the next free memory location. Then, after a delay, the copy springs to life, and itself makes a copy. Finally the whole of memory will be filled by copies of the program, and from the time taken to return to the monitor one can estimate the number of generations that lived.

Message

```
                        ; Displays a moving message on the
                        ; 7-segment displays
                        ; (Relocatable)
                        ;
0000                              .=OF1F
OF1F                    Speed:    .=.+1
                        ;
OF20    C40D            Tape:     LDI      H(Disp)
OF22    35                        XPAH     1
OF23    C400                      LDI      L(Disp)
OF25    31                        XPAL     1
OF26    C40F                      LDI      H(Text)
OF28    36                        XPAH     2
OF29    C4CA                      LDI      L(Text)-8
OF2B    32                        XPAL     2
OF2C    C4C0            Move:     LDI      X'C0      ;Determines sweep speed
```

```
0F2E    C8F0              ST      Speed
0F30    C407     Again:   LDI     7
0F32    01       Loop:    XAE
0F33    C280              LD      -128(2)
0F35    C980              ST      -128(1)
0F37    C4FF              LDI     X'FF
0F39    02                CCL
0F3A    70                ADE                 ;i.e. decrement ext.
0F3B    94F5              JP      Loop
0F3D    B8E1              DLD     Speed
0F3F    9CEF              JNZ     Again
0F41    C6FF              LD      @-1(2)      ;Move letters
0F43    94E7              JP      Move        ;X'80=end of text
0F45    90DF              JMP     Go
                 ;
        0D00     Disp     =       0D00
                 ;
                 ; A sample message
                 ; Message is stored backwards in memory
                 ; first character is 'end of text', X'80.
                 ; For a continuous message, first and
                 ; Last seven characters must be the
                 ; same (as in this case).
                 ;
0F47                      .=0FA0
0FA0                      .BYTE   080,079,079,06D,040,037
0FA6                      .BYTE   077,039,040,03E,0BF,06E     3F
0FAC                      .BYTE   040,06D,077,040,06E,03E
0FB2                      .BYTE   07F,040,079,037,030,071
0FB8                      .BYTE   040,06E,038,038,03F,01F
0FBE                      .BYTE   040,077,040,06D,030,040
0FC4                      .BYTE   039,040,071,03F,040,06D
0FCA                      .BYTE   040,079,079,06D,040,037
0FD0                      .BYTE   077,039
        0FD2     Text     =  .                ;start of message

                          .END
```

Self-Replicating Program

```
                 ; Makes a copy of itself and then
                 ; executes the copy.
                 ; Only possible in a processor which permits
                 ; one to write relocatable code, like SC/MP
                 ;
        FFFC     LDX      =       Loop-Head-1 ;offset for load
        000D     STX      =       Last-Store-1 ;offset for store
                 ;
0000                      .=0F12
0F12    C4FC     Head:    LDI     LDX
0F14    01                XAE
0F15    C080     Loop:    LD      -128(0)     ;PC-relative-ext=offset
```

```
0F17   01               XAE
0F18   02               CCL
0F19   F411             ADI    STX-LDX
0F1B   01               XAE
0F1C   C880    Store:   ST     -128(0)      ;ditto
0F1E   40               LDE
0F1F   03               SCL
0F20   FC10             CAI    STX-LDX-1    ;i.e. increment ext.
0F22   01               XAE
0F23   40               LDE
0F24   E414             XRI    Last-Loop-1  ;finished?
0F26   9CED             JNZ    Loop
0F28   8FFF             DLY    X'FF         ;shows how many copies
0F2A           Last     =                   ;were executed.
       0000             .END
```

Reaction Timer

```
                ; Gives readout of reaction time in milliseconds
                ; display lights up after a random delay
                ; Press'MEM' as quickly as possible.
                ; Press '0' to play again. (Relocatable)
                ; 150 = excellent, 250 = average, 350 = poor
                ;
       01E4     Cycles   =      500         ;SC/MP cycles per msec
       0F00     Ram.     =      0F00
       0D00     Disp     =      0D00
       0005     Adlh     =      5
       000C     Adl      =      12
       000E     Adh      =      14
       015A     Dispa    =      015A        ;'Address to segments'
                ;
0000                     .=0F20
0F20   C401     Begin:   LDI    H(Dispa)
0F22   37                XPAH   3
0F23   C459              LDI    L(Dispa)
0F25   33                XPAL   3
0F26   C205              LD     Adlh(2)     ; 'Random' number
0F28   01       Wait:    XAE
0F29   8F7D              DLY    Cycles/4
0F2B   02                CCL
0F2C   70                ADE                ;Count down
0F2D   94F9              JP     Wait
0F2F   C903              ST     +3(1)       ;Light'8' on display
0F31   40                LDE                ;Now zero
0F32   CA0C              ST     Adl(2)
0F34   CA0E              ST     Adh(2)
                ;Main loop ; length without DLY = 151 µcycles
0F36   C4A8     Time:    LDI    (Cycles-151—13)/2
0F38   8F00              DLY    0
0F3A   03                SCL
0F3B   C20C              LD     Adl(2)
```

```
0F3D  68                DAE
0F3E  CA0C              ST      Adl(2)
0F40  C20E              LD      Adh(2)
0F42  68                DAE
0F43  CA0E              ST      Adh(2)
0F45  40                LDE
0F46  02                CCL
0F47  F903              CAD     +3(1)    ;Test for key
0F49  98EB              JZ      Time
0F4B  3F        Stop:   XPPC    3        ;Go display time
0F4C  90FD              JMP     Stop     ;Illegal return
0F4E  90CF              JMP     Begin    ;Number key
                ;
0F50                    . = 0FF9         ;Pointers restored
                ;                        ;From ram
0FF9  0D00              .DBYTE  Disp     ;P1-> Display
0FFB  0F00              .DBYTE  Ram      ;P2-> Ram
      0000              .END
```

Science of Cambridge Limited
6 King's Parade
Cambridge CB2 1SN
Telephone Cambridge (0223) 311488